HOPLITE
SYSTEMS, LLC

Cognitive Processing Hardware Elements Final Technical Report

February 16, 2005

Sponsored by
Defense Advanced Research Projects Agency (DOD)
(DARPA/IPTO)

ARPA Order S016-37
Issued by U.S. Army Aviation and Missile Command Under
Contract No. W31P4Q-04-C-R301

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

HOPLITE
SYSTEMS, LLC

Cognitive Processing Hardware Elements
Final Technical Report

Name of Contractor: **Hoplite Systems, LLC**
Principal Investigator: **Matthew Scarpino**
Business Address: **3900 White Settlement Rd #183**
Fort Worth, TX 76107
Phone Number: **(817) 626-2454**
Fax Number: **(817) 378-8081**

Effective Date of Contract: **19 JUN 2004**
Short Title of Work: **Cognitive Processing Hardware Elements: Development of a Reconfigurable Architecture for Improved Soar Execution (RAISE)**

Contract Expiration Date: **18 FEB 2005**
Reporting Period: **JUN 2004 – FEB 2005**

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
16-02-2004	Final Technical Report	JUN 04 - FEB 05

4. TITLE AND SUBTITLE
Cognitive Processing Hardware Elements
Reconfigurable Architecture for Improved Soar Execution (RAISE)

5a. CONTRACT NUMBER
W31PQ-04-C-R301

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Matthew Scarpino, Hoplite Systems, LLC
Dr. Robert Wray, Soar Technologies

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Hoplite Systems, LLC
3900 White Settlement Rd,#183
Fort Worth, TX 76107

8. PERFORMING ORGANIZATION REPOR NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Sponsored by:
Dr. Robert Graybill
DARPA/IPTO
3701 North Fairfax Drive
Arlington, VA 22203-1714

Monitored by:
Thomas G. Bramhall
US Army Aviation/Missile Command
Building 7804, Room 222
Redstone Arsenal

10. SPONSOR/MONITOR'S ACRONYM(S)
Sponsor: DARPA/IPTO
Monitor: AMSRD-AMR-WS-DP-S

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

With help from Soar Technology, rule processing has been compared with both FPGAs and x8086 processors. For regular combinatorial logic, FPGAs present a speed-up of over x14000. However, they are presently unable to perform associative memory operations (pointer logic) that many rules require to operate. This shortcoming may be resolved with a dedicated memory management unit.

The proof-of-concept RAISE system has been successfully designed and demonstrated. It uses a Virtex-II FPGA to implement a minor Soar application. A GUI allows users to send Working Memory Elements (WMEs) through a serial connection and receive results from the operators. The source code for the FPGA and GUI are in Appendices A and B.

The first part of the Hoplite Guide to Run-time Reconfigurable Computing has been made available for free. Many have downloaded it and all comments have been positive.

15. SUBJECT TERMS
Cognitive Porcessing, Reconfigurable Computing, Field Programmable Gate Arrays, Java, JBits

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON Matthew Scarpino
a. REPORT UNCLASSIFIED	b. ABSTRACT UNCLASSIFIED	c. THIS PAGE UNCLASSIFIED	SAR	68	19b. TELEPHONE NUMBER *(include a code)* (817) 626-2454

TABLE OF CONTENTS

1. Summary

Hoplite's efforts during this initial work have focused on three goals: demonstrating the advantage of using FPGAs for cognitive processing, building a JBits application for simple FPGA-based Soar processing, and releasing a guide for JBits usage. Hoplite has successfully performed each of these tasks.

To show the merits of FPGA-based cognition, Hoplite performed rule processing with a Virtex-II FPGA and compared the timing results to those obtained from a traditional CPU. Soar Technology assisted by providing a set of files, with rule numbers ranging from 500 to 5000. Hoplite created an application to convert these Soar files into VHDL, and this code is presented in Appendix A. Each hardware design was used to configure a Xilinx' XC2V6000 FPGA and the timing results for each set of rules was measured. By keeping track of the maximum pin delay for each design, Hoplite determined how much time was needed to fire each rule. On average, FPGA-based rule processing provides a theoretical speed-up of 14000x and an experimental improvement of 6200x. The full results are presented in the Results section.

During this period, Hoplite has also designed and successfully implemented a complete Soar application, containing rules, operators, and preferences. In this application, the user selects the initial volumes of two water jugs and a desired volume. In regular Soar operation, the processing performs one of three operations – empty, fill, or pour – on each of the jugs until the desired value is met. But using JBits, we've built sub-circuits for each of these operations, and have configured the FPGA to perform them in parallel. We've also developed routines to send and retrieve data through a serial connection.

The third goal involves releasing a suitable JBits manual. In early September, we announced the first installment of the *Hoplite Guide to Run-time Reconfigurable Computing*, which can be downloaded at www.hopsys.com/whitepaper.html. The current document contains 79 pages including the copyleft license. However, there is still more to be added, including use of BRAM resources (memory and multipliers), and the clock generation/distribution capabilities.

2. Introduction

The theory and operation of the Reconfigurable Architecture for Improved Soar Execution (RAISE) is based on three concepts:

1. Rule Matching
2. Virtex-II FPGA operation
3. Run-time Reconfigurable Computing

The goal of this section is to provide an understanding of these topics. This will clarify the results achieved, and establish the vocabulary needed to express them.

2.1 Rule Matching

Cognitive modeling and expert systems differ in many respects, but both depend on rule matching to make decisions. This process begins with the perception system, which encodes external data into comprehensible information, called Working Memory Elements (WMEs). These elements are compared to sets of conditions, called rules, which make up the system's long-term memory. If elements in working memory match a rule's conditions, the rule will fire a production. [1]

For example, the following rule is taken from The Soar 8 Tutorial: [2]

```
sp {waterjug*propose*fill
    (state <s> ^name waterjug
               ^jug <j>)
    (<j> ^free > 0)
    -->
    (<s> ^operator <o> + =)
    (<o> ^name fill
         ^jug <j>)}
```

This rule has two conditions: that the current state be called `waterjug`, and that there is free space in the jug designated by `<j>`. If these are met, than the rule will recommend that the operator named `fill` should be used for jug `<j>`.

The more rules a cognitive system uses, the better it can react to external changes. But with a CPU, cycling through additional rules takes additional time, and system performance degrades. The Rete algorithm was designed to optimize rule matching, and it begins by decomposing WMEs into individual alpha memories according to the set of rule conditions. Then, it joins alpha memories together into beta memories, and the final beta nodes fire productions for the system. This is shown in Figure 1. With dynamic memory allocation, Rete can accommodate as many WMEs and rules as the computer's memory can hold. [3]

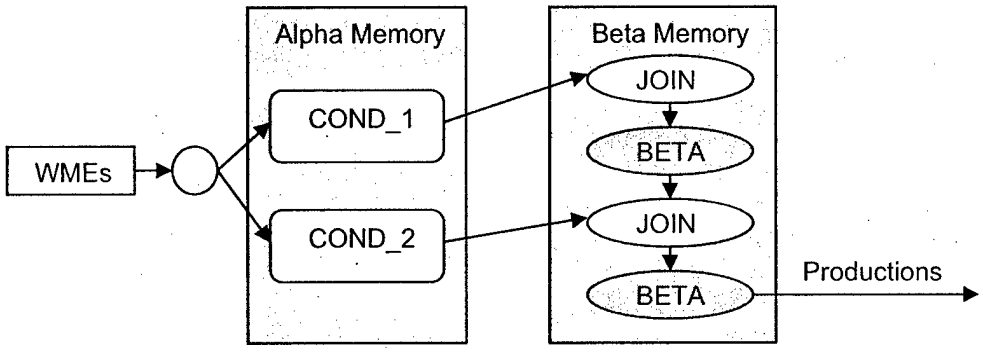

Figure 1: Simple Rete Structure

Rete is the optimal choice for matching rules with serial processors. However, performance could be improved further with Field Programmable Gate Arrays (FPGAs), which can process data in parallel. For this reason, Hoplite Systems has proposed implementing rule matching with these devices. The next section will show how they work.

2.2 Virtex-II Field Programmable Gate Arrays (FPGAs)

Entire books have been written on FPGA structure, but only two aspects are crucial for rule matching. First, it's important to understand how Virtex-II function generators perform boolean computation. Then, these results can be combined with sum-of-products chains.

2.2.1 Function generators and Look-up Tables (LUTs)

The basic computational unit of a Virtex-II FPGA is the Configurable Logic Block (CLB). This consists of four slices, and each slice contains two logical resources called *function generators*. Depending on how you configure them, they can serve as memory elements, shift registers, or look-up tables. For rule matching, we'll use them as look-up tables, or LUTs.

Virtex-II LUTs receive four separate inputs, and return a single output according to a logical function. You can use traditional functions like `AND` and `OR`, or you can create a completely custom relationship between the inputs and outputs. For rule matching, function generators can accept WMEs as inputs, combine them with a suitable boolean operation, and then return the result as a production or as a signal to feed into further logical stages.

To see why LUTs are so useful in rule matching, it's important to understand how many of them are available within an FPGA. Table 1 lists the different FPGAs available in the Virtex-II family, and shows how many resources are available in each. [4]

Table 1 Number of Function Generators available in Virtex-II FPGAs

FPGA	Configurable Logic Blocks	CLB Slices	Function Generators
XC2V40	64	256	512
XC2V80	128	512	1024
XC2V250	384	1536	3072
XC2V500	768	3072	6144
XC2V1000	1280	5120	10240
XC2V1500	1920	7680	15360
XC2V2000	2688	10752	21504
XC2V3000	3584	14336	28672
XC2V4000	5760	23040	46080
XC2V6000	8448	33792	67584
XC2V8000	11648	46592	93184

Since each function generator can receive four inputs, these devices can receive thousands of single-bit WMEs and perform their matching operations *at the same time*. Unlike regular CPUs, which perform only one operation per clock cycle, each function generator in an FPGA can operate independently. This means you don't need Rete to optimize serial processing – FPGA rule matching can be accomplished in parallel.

Most WMEs contain more than four bits of information, so it's necessary to combine LUTs to perform boolean operations on a larger scale. The Virtex-II designers made this possible by providing a sum-of-products chain.

2.2.2 Sum-of-products chains

The process of rule matching consists of boolean operations that combine elements from working memory and long-term memory. The result is a production. With boolean minimization, these operations can be implemented with FPGAs using as few LUTs as possible. There are many different algorithms for boolean minimization, but the result generally takes the same form: a sum-of-products equation. An example is shown below:

$$PROD = W_1W_4W_6 + W_3W_2 + W_1W_5$$

In this case, we have six single-bit WMEs ($W_{1\text{-}6}$) that combine to set the value of a production (PROD). AND gates are used for the multiplication and OR gates are used for addition. Every boolean relationship between WMEs and productions can be represented with a set of similar equations.

CLB slices contain dedicated circuitry to calculate sums of products. The LUTs (GLUT and FLUT) calculate the individual terms, and the ORCY gates provide the addition. These chains run between the slices in a CLB, and between different CLBs. This is shown in Figure 3. [5]

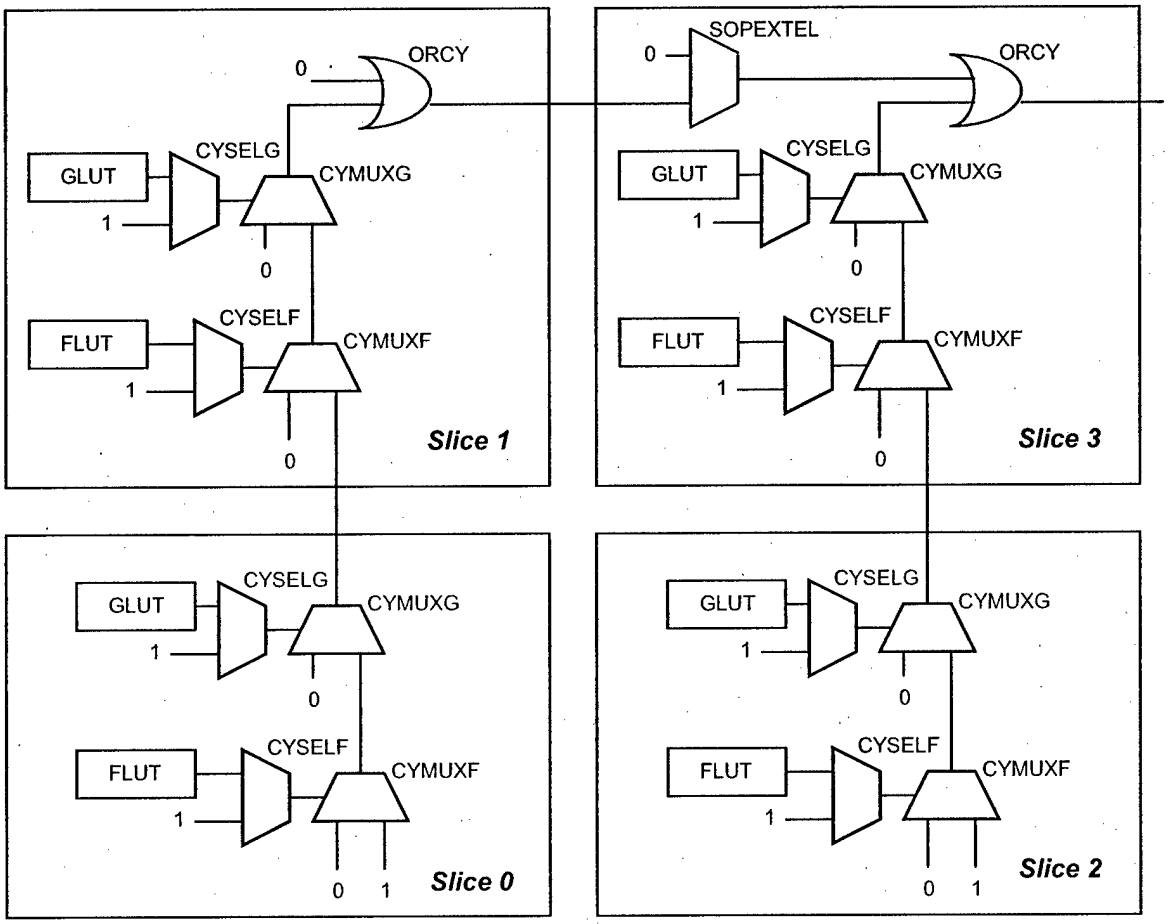

Figure 2: Sum-of-products circuitry within a Virtex-II FPGA

These circuits give FPGAs an important advantage when it comes to rule matching. But there is also one important disadvantage – FPGA designs are generally static. When you create a traditional FPGA design with VHDL or Verilog, you can't dynamically update working memory or the rules in long-term memory. But, because Xilinx has released the JBits library, it is possible to reconfigure the device during its operation. This process is called run-time reconfigurable computing.

2.3 JBits and Run-time Reconfigurable Computing

JBits is the first tool that allows digital designers to precisely control resources in an FPGA. This allows applications to be created that configure FPGAs for specific goals, such as rule processing. Further, designers can control the timing of the circuit by suitably routing internal connections. But the most important advantage of JBits is that it provides *partial reconfiguration.*

As mentioned above, FPGA designs are static. Once you've configured a CLB with a sum-of-products equation, you can't change it without reconfiguring the device. Usually, this means stopping processing and transferring a new bitstream into the FPGA. But with JBits, you can configure specific parts of the Virtex-II while other processing continues normally. This partial reconfiguration doesn't make FPGAs as dynamic as CPUs. But it reduces the difficulty of reconfiguring the device when a change needs to be made to working or long-term memory.

3. Methods, Assumptions, and Procedures

Our most important task during the reporting period involved comparing the rule processing efficiency of FPGAs with that of traditional x8086 architectures.

To support the use of FPGAs in cognitive processing, Hoplite has created a JBits application that uses Soar rules to perform the Waterjugs algorithm. After building circuits for the individual operators, we combined them to perform large-scale rule matching in parallel.

We've also released the Hoplite Guide to Run-time Reconfigurable Computing on our web site, at www.hopsys.com/whitepaper.html. This is the best documentation available on JBits, and we hope this will encourage others to use this important tool.

3.1 FPGAs versus Rete: Timing Comparison

Rule matching is the primary bottleneck in cognitive processing, and any innovation that can reduce the amount of time needed would be very helpful. This section shows how Hoplite compared the capabilities of FPGAs versus the Rete rule-matching algorithm. The results of this procedure are described in the Results section of this report.

The test procedure consists of three steps:

1. Rule Creation – SoarTech devised ten rule sets that focus on rule matching

2. Soar/VHDL Translation – Hoplite translated each rule file into an FPGA hardware design
3. FPGA Implementation and Testing – Each design was downloaded to an FPGA and tested

This section will explain each of these steps in detail, and will show many of the rules associated with the experiment.

3.1.1 Rule Creation

Throughout this effort, Hoplite has relied on Soar Technology (SoarTech) to provide assistance with the theory and software relating to the Soar cognitive model. Their primary activity involved creating sets of rules for implementation on FPGAs and CPUs. These rules don't encompass the entirety of Soar operation. Instead, they focus on generating productions based on Working Memory Elements (WMEs). In this manner, the test ensures that only rule-firing will be compared.

In addition to creating the rule files, SoarTech also tested the timing for each on a Pentium microprocessor. This provides a baseline against which to judge the merits of FPGA-based rule processing.

SoarTech designed a simple application that simulates the "birth" and "death" of members of ten families over the course of some number of generations. In order to avoid exponential memory growth (as one would expect with a generational model), the number of children per generation was limited to 10. This arrangement provides a near-constant memory-size after some initial bootstrapping, but leads to a large number of memory changes as new generations are "born" and old ones "die". Figure 3 illustrates the contents of memory during execution (the specific example comes from 500.soar).

This example was designed to provide a thorough investigation of the rule matching process. The following priorities were kept in mind:

- *Large rule sets*: Typical Soar applications range from 500-2000 rules. SoarTech generated rules sets from 500 to 5000 rules in 500-rule increments to form 10 distinct rule sets for experimental testing.
- *Large number of total conditions*: About half of the rules test for the name of an individual in the family tree. Thus, the number of unique conditions in a rule set is at least half that of the rules (250-2500 unique conditions).
- *Large number of memory changes*: The combination of a large number of rules and a large number of conditions doesn't ensure a large number of memory changes. SoarTech designed the application so that every rule in the rule set fires at least once, and that, after an initial bootstrapping period, within any match-fire cycle, many individual additions and deletions will be made to memory. This ensures a large number of memory changes.
- *Significant parallelism in the rules*: SoarTech designed the rule set to exhibit several orders of magnitude greater parallelism than found in typical Soar systems (200-600 production matches/cycle, vs. ~2-3 for a typical Soar system).
- *Focus on the Rete matcher*: The rules are designed to all fire within a single Soar decision cycle. This ensures that Soar's operator decision procedure and impasse mechanism are not used in the course of execution. For time-based measures, SoarTech

used Soar's DETAILED_TIMING_STATS compile-time flag in order to obtain time's reflecting the Rete match/fire execution cost for the execution of the 10 individual rule sets.

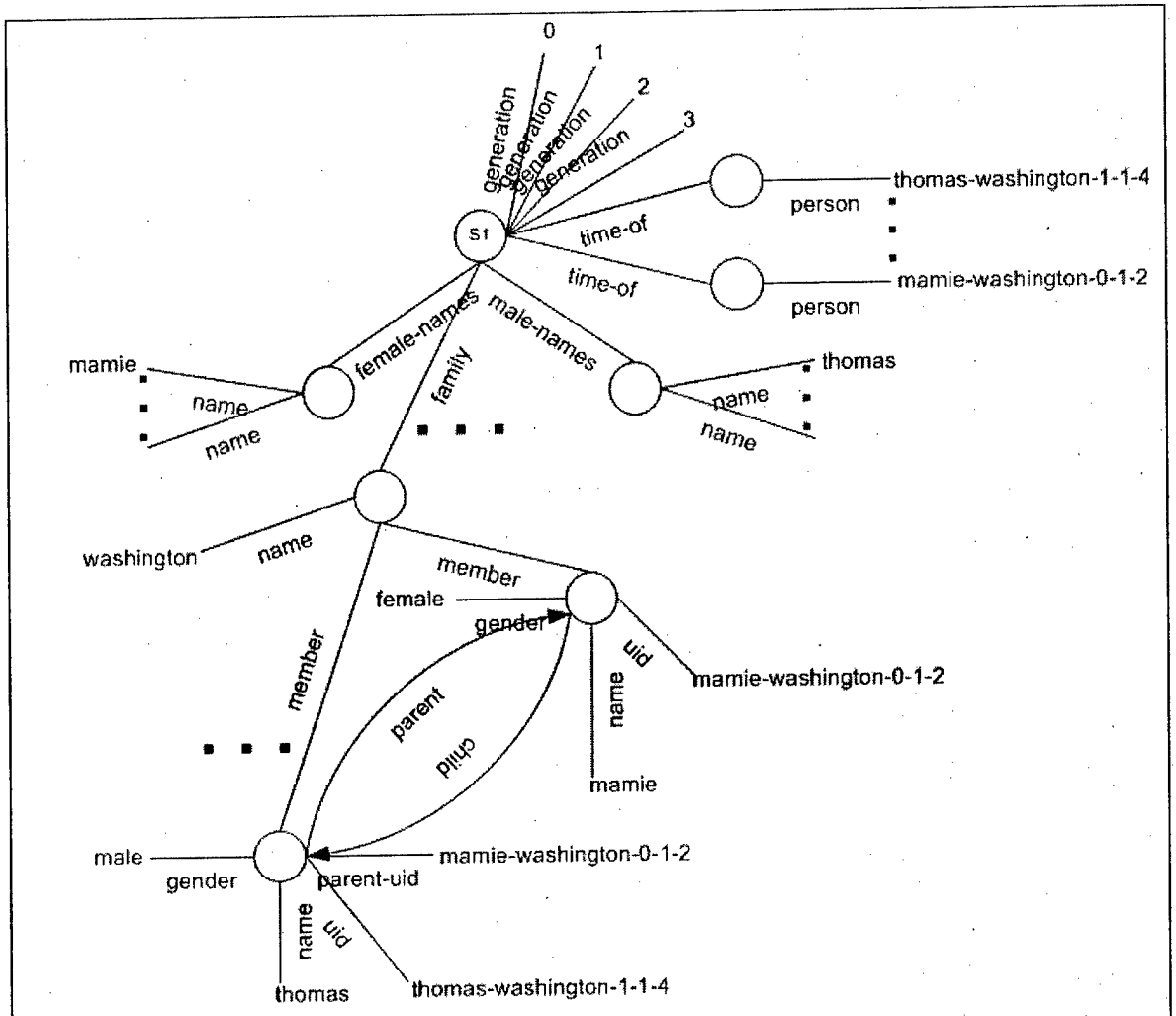

Figure 3: A partial view of the state generated by the test rule sets.

Table 2 lists the classes of productions in each of the rule sets generated for these experiments. Because names can be repeated within a family, each family member is identified with a "unique identifier" (UID). The UID consists of the full name (first name and last name) and three numbers. The first number indicates the generation of the family member. The second indicates the number of this child in this generation. Because there are ten children per generation, this number will be between 1 and 10. The last number indicates the production number for the UID production. For example, mamie-washington-0-1-2 indicates that Mamie Washington is the head of this family (0^{th} generation), is the first child (by default), and the production mamie-washington-0-1-2 is the second production generated (time-of-mamie-washington-0-1-2 is the first).

Table 2: Summary of productions in the example files.

Production Name/class	Description
create*families	Creates 10 "family" objects in memory. Figure 1 shows only one family, that of "Washington".
create*male-names	Creates an object "male-names" that holds all the possible male name in the examples. In Figure 1, "thomas" is listed as a male name.
create*female-names	Creates an object "female-names" that holds all the possible female name in the examples. In Figure 1, "mamie" is listed as a female name.
initial*generation	Set the initial generation to 0.
next-generation	Increment the generation. On each match-fire cycle, Soar will add a new generation value to the state. Previous values are not deleted because deletions would require invoking Soar's mechanism of persistence.

time-of-UID
Example:
`sp {time-of-thomas-washington-1-1-4`
`(state <s> ^time-of <t>`
`-^generation > 4`
`^generation >= 1 )`
`-->`
`(<t> ^person thomas-washington-1-1-4)}`
	These rules define the generations during which some family member should be "alive" (asserted). The example shows that the "time of" Thomas Washington is between generations 1 and 4 inclusively. In Figure 1, because there is no generation greater than 4, and there is a generation greater than 1, this rule is asserted, resulting in the addition of *thomas-washington-1-1-4* to memory. There are (total rules – 8)/2 of these rules.

UID
Example:
`sp {mamie-washington-0-1-2`
`(state <s> ^time-of <t>`
`^family <f>)`
`(<t> ^person mamie-washington-0-1-2)`
`(<f> ^name washington)`
`-->`
`(<f> ^member <m> +)`
`(<m> ^name mamie +`
`^uid mamie-washington-0-1-2 +)}`
	Asserts UID as a member of a particular family when the corresponding UID appears under "time-of". There are (total rules – 8)/2 of these rules.

General Inference Rules	
add-gender-of-child	Asserts a gender for each new family member, based on the list of male and female names.
add-pointer-to-parent	Makes a link (pointer) from a child to the corresponding parent (as specified by the parent-uid on the child). In Figure 1, there is a parent pointer from Thomas to Mamie.
add-children-of-parent	Makes a link (pointer) from a parent to the corresponding child (as specified by the parent pointer). In Figure 1, there is a child pointer from Mamie to Thomas.

To give a better idea of what these rules entail, Appendix A presents `500prods.soar`, which asserts and de-asserts a series of agents. Soar Technology provided this file and nine others with increasing numbers of rules. In addition, SoarTech ran these rules using the Soar kernel in general and the Rete algorithm in particular. The timing values for each rule set are presented in the Results section.

To process these rules on an FPGA, Hoplite needed a means of converting these rule files into designs that can be configured onto a Virtex-II device. To make this possible, a Java class was created to read in the *.soar files and output a *.vhd file. The next section presents how this process was performed.

3.1.2 Soar/VHDL Translation

The second file presented in Appendix A is `Convert.java`. This file reads in a Soar file and examines its variables, parameters, and attributes. Then, it uses this information to create a hardware design that can configure a Field Programmable Gate Array (FPGA). Because this is the first time that Soar rules have been processed on an FPGA, it's important to see how this was accomplished.

The first step involves determining the different values that each attribute can take. For example, there are ten different values that the LAST_NAME attribute can take. The conversion application uses this value to set the number of bits needed to contain this attribute. For the specified Soar rules, the bit allocation is presented as follows:

- *First name*: 72 possible (7 bits)
- *Last name*: 10 possible (4 bits)
- *Generation*: 24 possible (5 bits)
- *Step*: 19 maximum (5 bits)
- *Production Number*: 5000 possible (13 bits)

This bit minimization is an important difference from CPU operation, which uses a pre-selected number of bits (16, 32, or 64) for each variable. The advantage is that FPGA attributes take up the minimum amount of bits needed for processing. The disadvantage is that, when variable attributes require more memory, FPGAs can't replicate a CPU's dynamic allocation. This shortcoming is discussed in greater depth in the Conclusions section.

Once the conversion application determines how much memory is needed for each variable and working memory element, it creates a series of concurrent conditional statements in VHDL. While Soar rules are listed separately, these statements combine every rule that relates to a given production. This can be seen in the `prods500.vhd` file in Appendix A. An example is presented here:

```
with intime select
  WASHINGTON_asserted <=

  MAMIE & WASHINGTON & GEN0 & STEP1 & "000000010" when GEN0 & STEP1,
  THOMAS & WASHINGTON & GEN1 & STEP1 & "000000100" when GEN1 & STEP1,
```

This statement assigns the signal, `WASHINGTON_asserted`, based on a series of bit vectors. In Soar, this would take two rules to express, and therefore, two cycles to execute. However, this VHDL statement is concurrent, which means that the FPGA will check both statements at exactly the same time. To be more specific, this statement checks the value of `intime` to see whether it

equals `GEN0 & STEP1` or `GEN1 & STEP1`. If so, `WASHINGTON_asserted` is assigned to one of the values displayed at left of the `when` keyword.

As shown in Appendix A, all of the Soar rules can be implemented with these statements. The advantage is that each production can be performed in parallel, instead of each rule being performed separately. But to prove this advantage, Hoplite needed to implement the converted VHDL files onto a Virtex-II FPGA. The next section shows how this was accomplished.

3.1.3 FPGA Implementation and Testing

Once the VHDL files were created, Hoplite used them to create Soar circuits inside a Field Programmable Gate Array (FPGA). In particular, we used Xilinx' XC2V6000 FPGA, which is located on the board shown in figure 4. With over 6 million gates, this device has more than enough capacity to store all of the Soar rules in any given set.

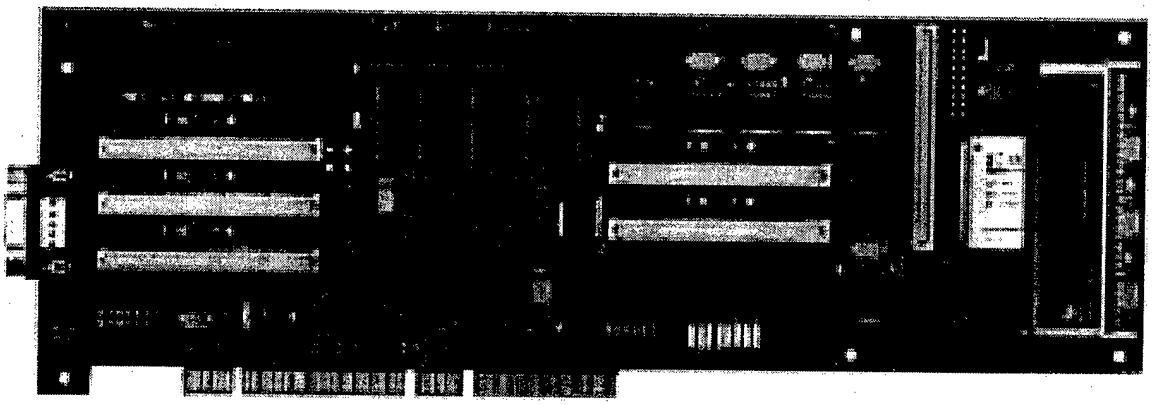

Figure 4: FPGA circuit board used to perform Rule Processing

The process of converting the VHDL file into a configuration bitstream was performed by the Xilinx Foundation ISE software. This entails three steps. First, the VHDL file is checked for errors in syntax and bit allocation. During the synthesis stage, it is converted into a device-independent logic file called a netlist. The final stage, implementation, converts the netlist into a large bit array to be downloaded to the device.

During implementation, the Xilinx software determines how much time will be needed for each signal to transfer through the circuit. It also calculates the maximum time needed for the input signals to reach the output. This value, called the maximum pin delay, provides a means of determining the maximum amount of time needed for each clock cycle. When a clock cycle has a period larger than this delay, then each output will have had enough time to reach a stable value.

To determine how much time is needed for the FPGA to perform a given rule set, the first step involves finding how many agents need to be created. This tells how many clock cycles are required to create each agent in the rule set. Then, this number of cycles is multiplied by the time needed per cycle. The results for these rule set computations are presented in the Results section, on page 22.

3.2 The Soar Waterjugs Algorithm on an FPGA

Comparing FPGA and Retc rule processing has been our primary goal, but we felt is was important to build a proof-of-concept Soar/FPGA application that incorporates operators as well as productions. A common problem in cognitive processing involves pouring, filling, and emptying water in two jugs (<i> and <j>) until a desired volume is reached. We began by building circuits for each of the Soar operators – pour, fill, and empty. Then, we connected these operators to perform multiple rule matches at a time. We developed serial communication between a CPU and the FPGA through an RS-232 serial cable. Finally, we coded a GUI to initialize the WMEs and control the FPGA's operation.

3.2.1 Circuit design for Soar operators

The first step in the design process involves using JBits to build circuits for the different operators in the Soar Waterjugs application. They are as follows:

1. *Empty* – Set the contents of a non-empty jug to zero
2. *Fill* – Set the contents of a non-full jug to its full value
3. *Pour* – Transfer contents from a non-empty jug to a non-full jug

The Soar rules specify conditions for when these operators can be used. As listed in the Soar 8 Tutorial, they are: [2]

```
sp {waterjug*propose*fill
    (state <s> ^name waterjug
               ^jug <j>)
    (<j> ^free > 0)
    -->
    (<s> ^operator <o> +
    (<o> ^name fill
         ^jug <j>)}

sp {waterjug*propose*empty
    (state <s> ^name waterjug
               ^jug <j>)
    (<j> ^contents > 0)
     -->
    (<s> ^operator <o> + =)
    (<o> ^name empty ^jug <j>)}

sp {waterjug*propose*pour
   (state <s> ^name waterjug
              ^jug <i> { <> <i> <j> })
   (<i> ^contents > 0 )
   (<j> ^free > 0)
   -->
   (<s> ^operator <o> + =)
```

```
(<o> ^name pour ^jug <i> ^into <j>)}
```

As shown, each jug has two values that set its state: `contents` and `free`. The first field specifies the volume of water in the jug, and the second tells how much of the jug is empty. Each operator, if activated, modifies both values. Since the maximum value of either field is seven, we've chosen to make both values three bits wide. Therefore, the state of a single jug consists of six bits.

Each of these three rules specifies a specific jug, but both jugs need to be evaluated. So, for each pair of waterjugs, we created a circuit of six sub-circuits. The fill and empty circuits were very easy to code, but the pour operator was more involved.

The Fill operator

The first operator in the Waterjugs application, Fill, is very simple to understand. First, compare the jug's `contents` field with its `full` value. If they're equal, leave the `contents` and `empty` fields unchanged. If not, change the `contents` value to `full` and the `empty` value to zero.

Figure 5 shows the logic circuit that makes this possible:

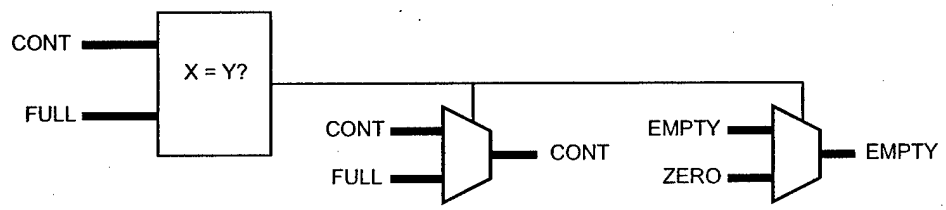

Figure 5: The circuit representing the Fill operator

This requires three CLBs in the Virtex-II. Slice 0 of the first CLB takes in the `contents` and `full` signals and returns a '1' if they're equal and a '0' if not. Slice 1 uses this result to set the first bit of the output `contents` and `empty` signals. The code for this CLB is given below:

```
jbits.setTileBits(2, 2, CY0F.CONFIG[0], CY0F.ZERO);
jbits.setTileBits(2, 2, CY0G.CONFIG[0], CY0G.ZERO);
jbits.setTileBits(2, 2, CYSELF.CONFIG[0], CYSELF.F);
jbits.setTileBits(2, 2, CYSELG.CONFIG[0], CYSELG.G);
jbits.setTileBits(2, 2, FLUT.CONTENTS[0], new int[]
  {0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0});
jbits.setTileBits(2, 2, FLUT.CONTENTS[1], new int[]
  {1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0});
jbits.setTileBits(2, 2, FXMUX.CONFIG[0], FXMUX.FXOR);
jbits.setTileBits(2, 2, GLUT.CONTENTS[0], new int[]
  {0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0});
jbits.setTileBits(2, 2, GLUT.CONTENTS[1], new int[]
  {1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0});
jbits.setTileBits(2, 2, GYMUX.CONFIG[0], GYMUX.ORCY);
jbits.setTileBits(2, 2, BX2.BX2, BX2.OMUX_E2);
jbits.setTileBits(2, 2, BY2.BY2, BY2.BX2);
```

```
jbits.setTileBits(2, 2, E6BEG4.E6BEG4, E6BEG4.Y1);
jbits.setTileBits(2, 2, F1_B0.F1_B0, F1_B0.N2MID7);
jbits.setTileBits(2, 2, F1_B1.F1_B1, F1_B1.Y0);
jbits.setTileBits(2, 2, F2_B0.F2_B0, F2_B0.S2MID4);
jbits.setTileBits(2, 2, F3_B0.F3_B0, F3_B0.N2MID6);
jbits.setTileBits(2, 2, F3_B1.F3_B1, F3_B1.W2MID4);
jbits.setTileBits(2, 2, F4_B0.F4_B0, F4_B0.W2MID0);
jbits.setTileBits(2, 2, F4_B1.F4_B1, F4_B1.BY2);
jbits.setTileBits(2, 2, G1_B1.G1_B1, G1_B1.Y0);
jbits.setTileBits(2, 2, G2_B0.G2_B0, G2_B0.W2MID4);
jbits.setTileBits(2, 2, G3_B1.G3_B1, G3_B1.OMUX_E7);
jbits.setTileBits(2, 2, G4_B0.G4_B0, G4_B0.OMUX_E2);
jbits.setTileBits(2, 2, N2BEG0.N2BEG0, N2BEG0.W2MID0);
jbits.setTileBits(2, 2, N2BEG8.N2BEG8, N2BEG8.X1);
jbits.setTileBits(2, 2, OMUX11.OMUX11, OMUX11.YB0);
jbits.setTileBits(2, 2, X0.X0, X0.CLB_XORF_S0);
jbits.setTileBits(2, 2, Y0.Y0, Y0.CLB_ORCY_S0);
```

The second and third CLBs in the Fill circuit determine the second and third bits of the `contents` and `empty` signals. In both cases, only the bottom slice is used. The code for these circuits is shown below:

```
jbits.setTileBits(4, 2, FLUT.CONTENTS[0], new int[]
  {1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0});
jbits.setTileBits(4, 2, GLUT.CONTENTS[0], new int[]
  {1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0});
jbits.setTileBits(4, 2, F1_B0.F1_B0, F1_B0.N2MID7);
jbits.setTileBits(4, 2, F3_B0.F3_B0, F3_B0.N2MID6);
jbits.setTileBits(4, 2, F4_B0.F4_B0, F4_B0.OMUX_E2);
jbits.setTileBits(4, 2, G1_B0.G1_B0, G1_B0.N2MID7);
jbits.setTileBits(4, 2, G2_B0.G2_B0, G2_B0.OMUX_E7);
jbits.setTileBits(4, 2, OMUX10.OMUX10, OMUX10.Y0);
jbits.setTileBits(4, 2, OMUX14.OMUX14, OMUX14.X0);

jbits.setTileBits(3, 2, FLUT.CONTENTS[0], new int[]
  {1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0});
jbits.setTileBits(3, 2, GLUT.CONTENTS[0], new int[]
  {1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0});
jbits.setTileBits(3, 2, F1_B0.F1_B0, F1_B0.N2END7);
jbits.setTileBits(3, 2, F3_B0.F3_B0, F3_B0.OMUX_N11);
jbits.setTileBits(3, 2, F4_B0.F4_B0, F4_B0.N2MID0);
jbits.setTileBits(3, 2, G2_B0.G2_B0, G2_B0.OMUX_E7);
jbits.setTileBits(3, 2, G3_B0.G3_B0, G3_B0.OMUX_N11);
jbits.setTileBits(3, 2, N2BEG6.N2BEG6, N2BEG6.N2END6);
jbits.setTileBits(3, 2, N2BEG7.N2BEG7, N2BEG7.OMUX_N11);
jbits.setTileBits(3, 2, N2BEG9.N2BEG9, N2BEG9.X0);
jbits.setTileBits(3, 2, OMUX14.OMUX14, OMUX14.Y0);
jbits.setTileBits(3, 2, S2BEG4.S2BEG4, S2BEG4.OMUX_SE3);
```

This is a straightforward circuit, and can be compressed into two CLBs if needed. The circuit for the Empty operator is even simpler.

The Empty operator

The Empty circuit is similar to the Fill circuit, except that instead of a comparator, you only need an OR gate to determine if all of the bits in `contents` are zero. If so, the `output` contents and `empty` signals remain as they are. If not, `contents` is set to zero and `empty` is set to full. Figure 6 shows the logic diagram for this circuit.

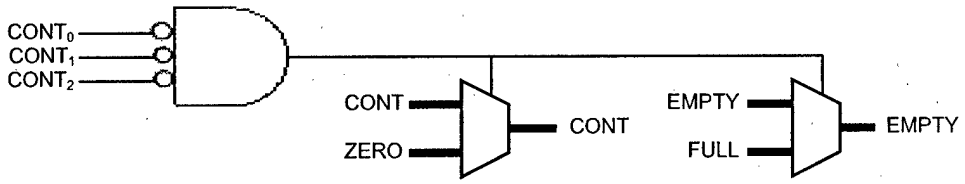

Figure 6: The circuit representing the Empty operator

Just as the logic of the Empty operator resembles that of the Fill operator, the JBits code for the two circuits is also similar. The main difference is that the inverted AND gate only needs a single slice. The code for this is shown below. Also, since the `contents` signal will be set to zero in either case, there's no need for a multiplexer.

```
jbits.setTileBits(2, 2, FLUT.CONTENTS[0], new int[]
  {1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0});
jbits.setTileBits(2, 2, FLUT.CONTENTS[1], new int[]
  {1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0});
jbits.setTileBits(2, 2, GLUT.CONTENTS[0], new int[]
  {1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0});
jbits.setTileBits(2, 2, GLUT.CONTENTS[1], new int[]
  {0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1});
jbits.setTileBits(2, 2, BY2.BY2, BY2.S2END4);
jbits.setTileBits(2, 2, F1_B0.F1_B0, F1_B0.BY2);
jbits.setTileBits(2, 2, F2_B0.F2_B0, F2_B0.Y1);
jbits.setTileBits(2, 2, F2_B1.F2_B1, F2_B1.W2MID5);
jbits.setTileBits(2, 2, F3_B1.F3_B1, F3_B1.Y1);
jbits.setTileBits(2, 2, F4_B0.F4_B0, F4_B0.W2END0);
jbits.setTileBits(2, 2, F4_B1.F4_B1, F4_B1.S2MID9);
jbits.setTileBits(2, 2, G2_B0.G2_B0, G2_B0.Y1);
jbits.setTileBits(2, 2, G2_B1.G2_B1, G2_B1.N2MID6);
jbits.setTileBits(2, 2, G3_B0.G3_B0, G3_B0.OMUX_SE3);
jbits.setTileBits(2, 2, G3_B1.G3_B1, G3_B1.OMUX_E7);
jbits.setTileBits(2, 2, G4_B0.G4_B0, G4_B0.W2MID0);
jbits.setTileBits(2, 2, G4_B1.G4_B1, G4_B1.N2MID7);
jbits.setTileBits(2, 2, OMUX1.OMUX1, OMUX1.X1);
jbits.setTileBits(2, 2, OMUX10.OMUX10, OMUX10.X0);
jbits.setTileBits(2, 2, OMUX14.OMUX14, OMUX14.Y0);
```

The Fill and Empty operators are easy to convert into FPGA resources, but the Pour operator is more involved. Not only does it deal with logic, it also requires addition and subtraction. The next section shows how this operator was constructed.

The Pour operator

The Pour operator transfers the contents of one jug to the other. The nature of the operation depends on the relative values of the `contents` field of the first jug and the `empty` field of the second jug. If `contents` is larger, then the Pour operation will fill the second jug, and the `contents` field of the first jug will be the difference between the two. If `empty` is larger, the first jug will be empty and the second jug will contain the sum of the two `contents` values.

The logical circuit for this operation is shown in Figure 7.

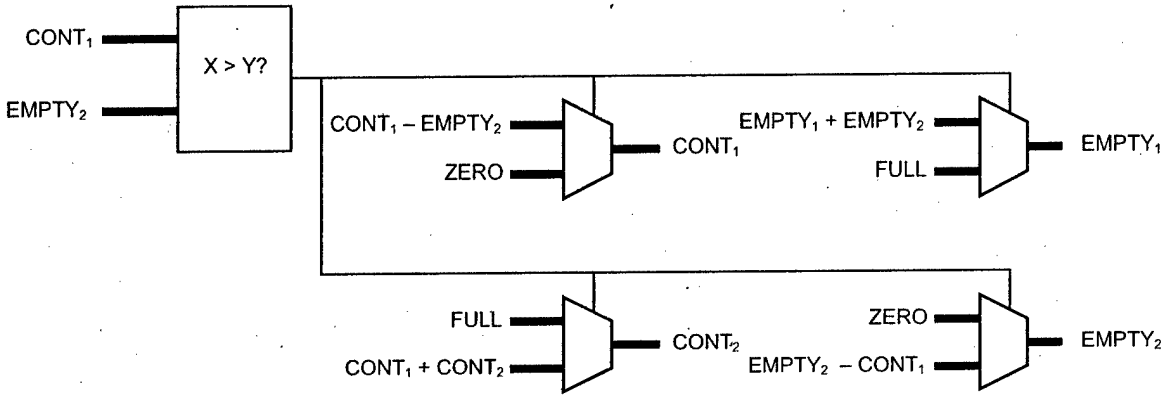

Figure 7: The circuit representing the Pour operator

The JBits code for this circuit runs for several pages. Since we've already covered the circuits for the three-bit comparator and multiplexer, we'll focus on addition and subtraction. Virtex-II FPGAs add and subtract using carry-lookahead circuits. Figure 8 depicts their structure. [5]

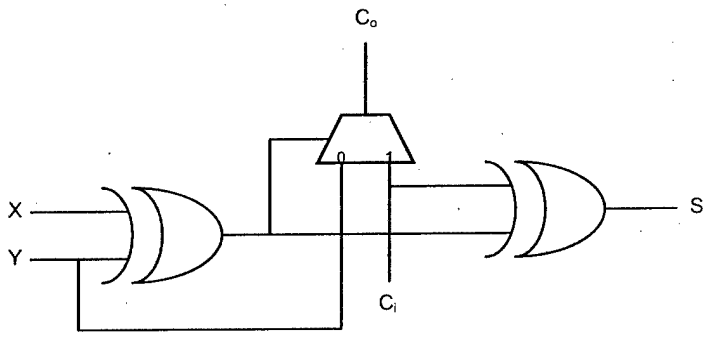

Figure 8: The carry-lookahead addition circuit

To create our adder circuits, we configured a function generator as an XOR gate and made the appropriate connections for the carry logic. This is shown in the following code:

```
jbits.setTileBits(4, 3, CY0F.CONFIG[0], CY0F.F1);
jbits.setTileBits(4, 3, CY0G.CONFIG[0], CY0G.G2);
jbits.setTileBits(4, 3, CYINIT.CONFIG[1], CYINIT.CIN);
jbits.setTileBits(4, 3, CYSELF.CONFIG[0], CYSELF.F);
jbits.setTileBits(4, 3, CYSELG.CONFIG[0], CYSELG.G);
jbits.setTileBits(4, 3, FLUT.CONTENTS[0], new int[]
```

```
  {1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1});
jbits.setTileBits(4, 3, FLUT.CONTENTS[1], new int[]
  {1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1});
jbits.setTileBits(4, 3, FXMUX.CONFIG[0], FXMUX.FXOR);
jbits.setTileBits(4, 3, FXMUX.CONFIG[1], FXMUX.FXOR);
jbits.setTileBits(4, 3, GLUT.CONTENTS[0], new int[]
  {1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1});
jbits.setTileBits(4, 3, GLUT.CONTENTS[1], new int[]
  {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1});
jbits.setTileBits(4, 3, GYMUX.CONFIG[0], GYMUX.GXOR);
jbits.setTileBits(4, 3, BX0.BX0, BX0.OMUX_E2);
jbits.setTileBits(4, 3, F1_B0.F1_B0, F1_B0.N2END7);
jbits.setTileBits(4, 3, F1_B1.F1_B1, F1_B1.E2MID0);
jbits.setTileBits(4, 3, F2_B0.F2_B0, F2_B0.E2MID4);
jbits.setTileBits(4, 3, F2_B1.F2_B1, F2_B1.N2END6);
jbits.setTileBits(4, 3, G1_B0.G1_B0, G1_B0.S2MID8);
jbits.setTileBits(4, 3, G2_B0.G2_B0, G2_B0.E2MID3);
jbits.setTileBits(4, 3, OMUX10.OMUX10, OMUX10.X1);
jbits.setTileBits(4, 3, OMUX14.OMUX14, OMUX14.Y0);
jbits.setTileBits(4, 3, W2BEG6.W2BEG6, W2BEG6.X0);
```

The subtraction circuit is nearly identical, except that the function generator logic is reversed. This is shown in the code shown below:

```
jbits.setTileBits(4, 3, CY0F.CONFIG[0], CY0F.F1);
jbits.setTileBits(4, 3, CY0G.CONFIG[0], CY0G.G2);
jbits.setTileBits(4, 3, CYINIT.CONFIG[1], CYINIT.CIN);
jbits.setTileBits(4, 3, CYSELF.CONFIG[0], CYSELF.F);
jbits.setTileBits(4, 3, CYSELG.CONFIG[0], CYSELG.G);
jbits.setTileBits(4, 3, FLUT.CONTENTS[0], new int[]
  {0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0});
jbits.setTileBits(4, 3, FLUT.CONTENTS[1], new int[]
  {0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0});
jbits.setTileBits(4, 3, FXMUX.CONFIG[0], FXMUX.FXOR);
jbits.setTileBits(4, 3, FXMUX.CONFIG[1], FXMUX.FXOR);
jbits.setTileBits(4, 3, GLUT.CONTENTS[0], new int[]
  {0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0});
jbits.setTileBits(4, 3, GLUT.CONTENTS[1], new int[]
  {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1});
jbits.setTileBits(4, 3, GYMUX.CONFIG[0], GYMUX.GXOR);
jbits.setTileBits(4, 3, F1_B0.F1_B0, F1_B0.N2END7);
jbits.setTileBits(4, 3, F1_B1.F1_B1, F1_B1.E2MID0);
jbits.setTileBits(4, 3, F2_B0.F2_B0, F2_B0.E2MID4);
jbits.setTileBits(4, 3, F2_B1.F2_B1, F2_B1.N2END6);
jbits.setTileBits(4, 3, G1_B0.G1_B0, G1_B0.S2MID8);
jbits.setTileBits(4, 3, G2_B0.G2_B0, G2_B0.E2MID3);
jbits.setTileBits(4, 3, OMUX10.OMUX10, OMUX10.X1);
jbits.setTileBits(4, 3, OMUX14.OMUX14, OMUX14.Y0);
jbits.setTileBits(4, 3, W2BEG6.W2BEG6, W2BEG6.X0);
```

The Pour operator circuit is constructed by combining these circuits with a comparator and multiplexers. The entire Waterjugs circuit is constructed by combining the Fill, Empty, and Pour circuits together. The next section explains how this is done.

3.2.2 Combining the Soar operators

The regular Soar Waterjugs application performs one operator (Fill, Empty, or Pour) per cycle. But because FPGAs can activate multiple circuits at once, we've chosen to perform all six operators per cycle. This is shown in Figure 9.

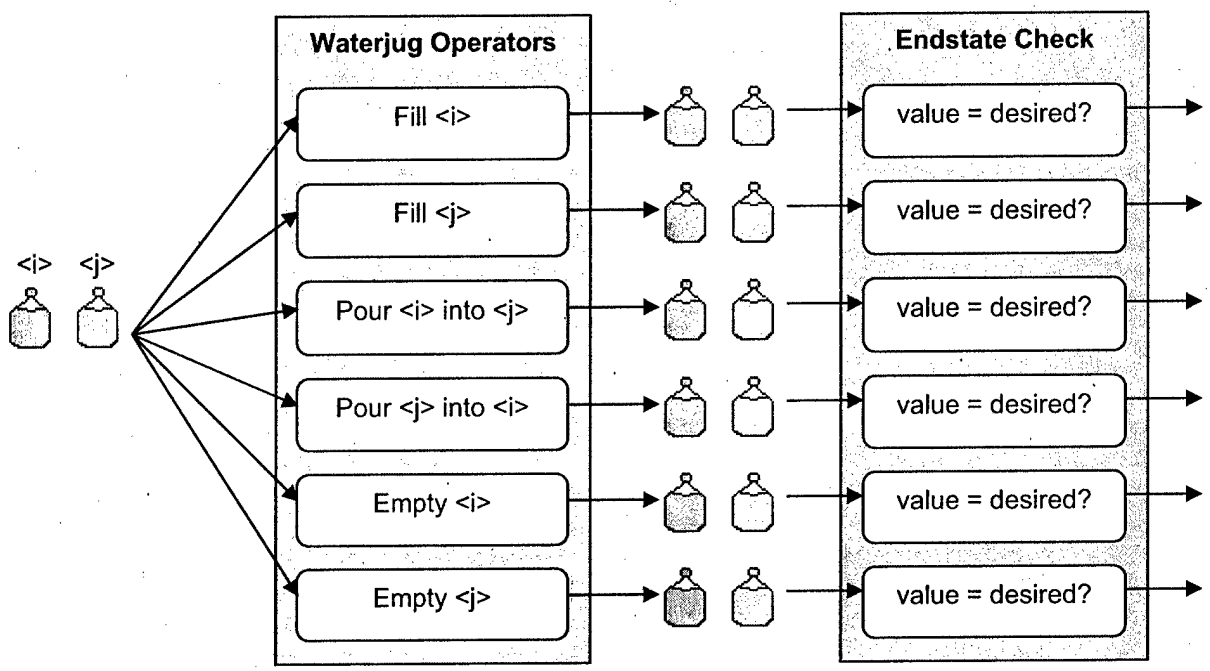

Figure 9: The initial Waterjug processing stage

Here, each pair of water jugs is converted into six pairs. Then, each of these pairs is compared to the desired value. If this value is reached, a '1' is sent. If not, further iterations are necessary.

When we first completed the circuit as shown, it used very little of the XC2V1000's resources. Also, for the initial conditions – Jug 1 has a volume of 5, Jug 2 has 3 – the operations need to be repeated 168 times on average. To better display the capabilities of the FPGA, we built a circuit that performs every permutation of each operator for four stages.

In the first stage, the initial pair becomes six pairs. Then, those six pairs become thirty-six pairs. In the third stage, there are 216 pairs of jugs that need to be checked for the desired volume. If it still hasn't been reached, the fourth stage performs the operator 1296 times, and checks each one to see if the desired value is matched. These stages are shown in Figure 9.

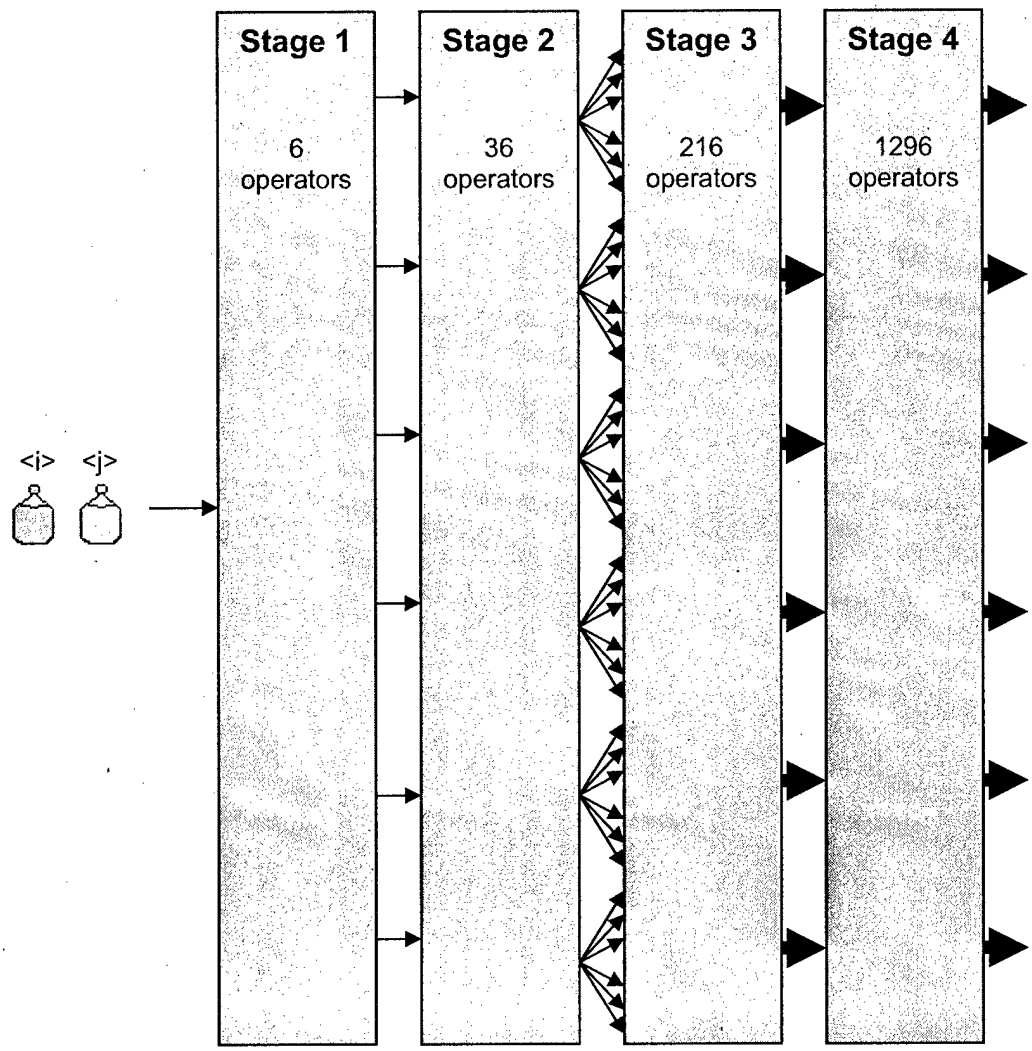

Figure 10: The complete Waterjug processing circuit

We used two simplifications to reduce the circuit size. First, if the jugs are full at the start, neither the fill nor pour operators will change the state. Then, unless the desired volume is zero or full, the final operator must be the pour operator.

The most difficult aspect of this design wasn't configuring the operator logic, but manually setting the connections needed to route signals between the CLBs. Appendix A presents the code for this complete design.

In addition to the operator circuits, we also implemented shift registers for input and output. The FPGA receives the volumes of the initial jugs and the desired value through a nine-bit shift register. Then, each stage holds a shift register that contains a '1' if the corresponding jug pair met the desired value, and a '0' if not.

A serial connection is needed to transfer the contents of these registers to and from the CPU. The software that makes this connection possible is part of the GUI, which we'll describe next.

3.2.3 The Graphical User Interface

A graphical application is needed to set the initial parameters and read the results of the FPGA's operation. For this purpose, we used IBM's Standard Widget Toolkit (SWT) to create the Java-based application shown in Figure 11.

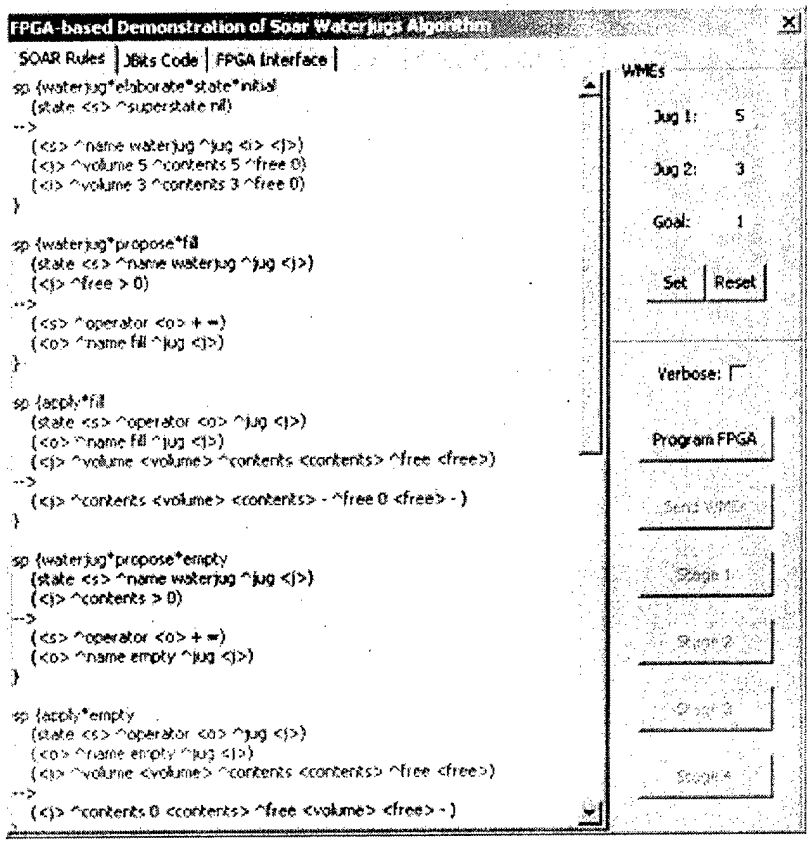

Figure 11: The Waterjug Graphical User Interface

The window on the left has three panes. The first shows the Soar rules that will be implemented onto the FPGA, the second shows the JBits code that will be used, and the third shows the results coming from the RS-232 connection. On the upper right, three text boxes control the algorithm's initial parameters – the volume of both jugs and the desired volume of the second jug.

The buttons on the right control the FPGA processing. The top button, "Program FPGA," sends the bitstream to the FPGA through the JTAG cable. The next button, "Send WMEs," transfers the initial parameters to the FPGA so that it can begin processing. The next four buttons, "Stage 1" through "Stage 4," tell the FPGA to direct its output through the serial port. These results are displayed on the "FPGA Interface" pane on the upper-left.

If the FPGA produces a '1' during any of the four stages, the GUI will display the operators that led to the desired result. If the 'verbose' box is checked, the GUI will also display operators that didn't produce successful results.

3.3 Hoplite Guide to Run-time Reconfigurable Computing

In September 2004, Hoplite released a JBits manual called the Hoplite Guide to Run-time Reconfigurable Computing. In its nearly eighty pages, it covers most of the theory needed for programming JBits and understanding FPGA structure. Chapter 1 provides a top-level introduction to JBits development. Chapter 2 focuses on Input/Output Blocks, and describes how they connect to the Global Routing Matrix (GRM). Chapter 3 describes the Configurable Logic Blocks (CLBs) in great depth. Example code is provided for each chapter.

4. Results

The primary experiment in this Phase I effort involved determining whether FPGAs present a suitable alternative to Rete when it comes to processing rules. Because there is very little way to measure efficiency values, we made this comparison using timing results from both activities. Soar Technologies performed rule processing with the Rete algorithm in the Soar kernel. Hoplite Systems performed rule processing with the VHDL files created from Soar's rule sets. The results of these experiments are presented below.

4.1 Soar Technology's Results with Rete Rule Processing

The Soar Rete results were obtained on a Dell Inspiron 8100 laptop (Pentium III (R) Mobile) at 1000MHz CPU clock speed and 512 M of RAM. The operating system was Windows 2000, 5.00.2195 (Service Pack 5). SoarTech used the following notation to refer to specific runs of the application: (# of rules, F | D). Thus, (500, F) refers to a complete 500-rule rule set; (4500,D) refers to the 4500-rule rule set with the expensive rules deleted. Each application was run until all family members had been asserted and retracted.

Figure 12 shows the relationship between the number of production fired in the course of execution, and the total match time for both the full rule sets (blue) and the rule set with the three expensive rules deleted (pink). Production firings increase monotonically with total rules, so the left-most data point in each line represents the 500-rule application, the next to the right the 1000-rule set, and so on. Because the expensive rules account for such a large number of production firings (each of the three rules will fire once for each family member in the population), production firings drop substantially in the second case. However, this graph makes clear the impact of the expensive rules. For example, both (2500,F) and (4000,D) generate around 10000 production firings, but the rule set with expensive rules consumes 8.7 times as much computation time as the larger rule set without the expensive rules. One goal of a parallel implementation of production matching should be to significantly decrease the cost of expensive matching, thus reducing the slope of the "Full System" line in these experiments.

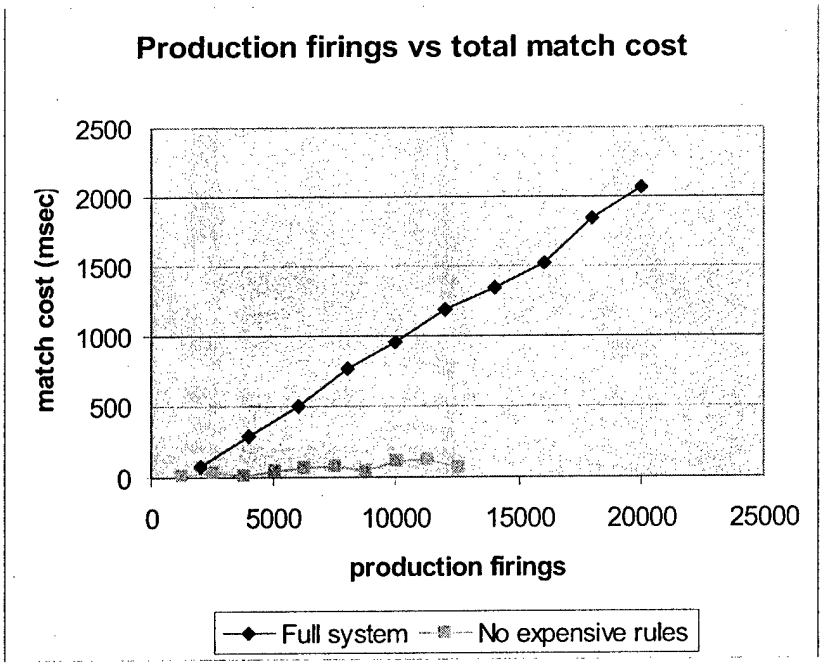

Figure 12: Productions firings vs. total match cost.

Figure 12 demonstrates that there is real cost for simulated parallelism in Soar's Rete implementation. In the complete rule-set case, as the average number of production firings increases about 3-fold (200 to 600), the average match cost per cycle increases by a factor of nearly 8. Although more difficult to see due to scaling, the rule set without expensive rules also increases, with a 3-fold increase in rule firings/cycle resulting in a 4-fold increase in average match time per cycle. While this increase is relatively smaller, in both cases, increased parallelism in the rules results in increased execution costs. For a truly parallel implementation, the resulting graph should be (nearly) flat, indicating that additional rules matched and fired within a cycle should not incur incrementally more elapsed time.

Figure 13 represents the same data from Figure 12 in a slightly different way. This graph plots the total number of match-fire cycles for each rule set vs. the average match time per cycle. As Figure 12 showed, in these rule sets, the number of rule firings per match-fire cycle goes up as the total size of the rule set goes up; the number of rule firings per match-fire cycle is shown for each point in parentheses. Because every match-fire cycle represents, in a parallel conception, a single "unit" of computation, the lines in this diagram ideally would be horizontal.

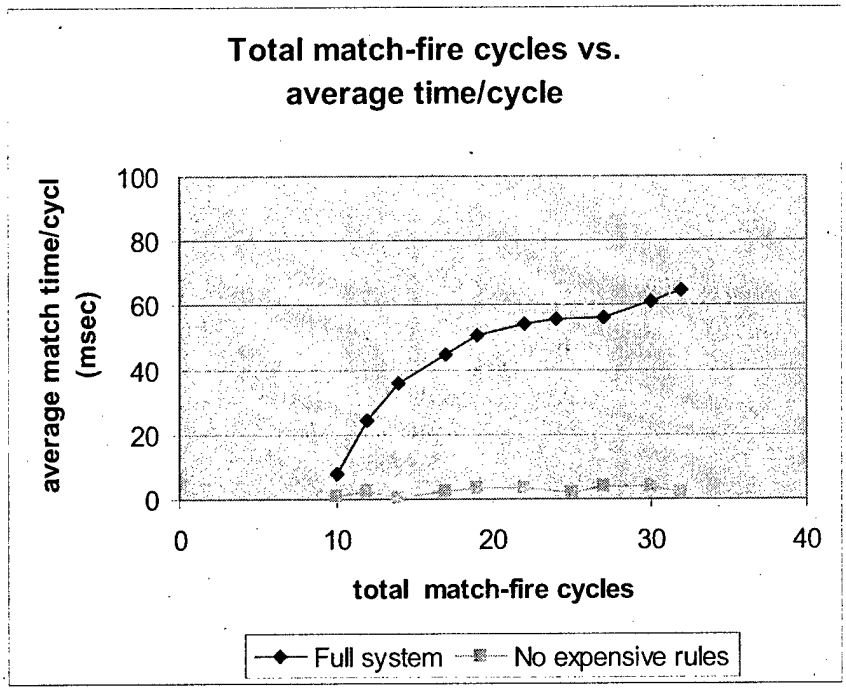

Figure 13: Total match-fire cycles vs. average time per cycle.

One final question to address is whether the positive slopes evident in Figure 13 and Figure 14 really reflect the cost of executing serial production firings in parallel. For example, it could be that other mechanisms in Soar are responsible for the increases in cost. To test this assumption, we executed all the rule sets for a fixed number of cycles. We chose 10 cycles. Figure 14 shows the results of this experiment, plotting the number of rules in the application vs. the total match time. The numbers in parentheses report the number of rule firings per match-fire cycle. As in the previous figures, as rule firings per cycle increases, the total match time also increases. However, for the rule sets above 1500 rules, the number of rule firings per cycle is

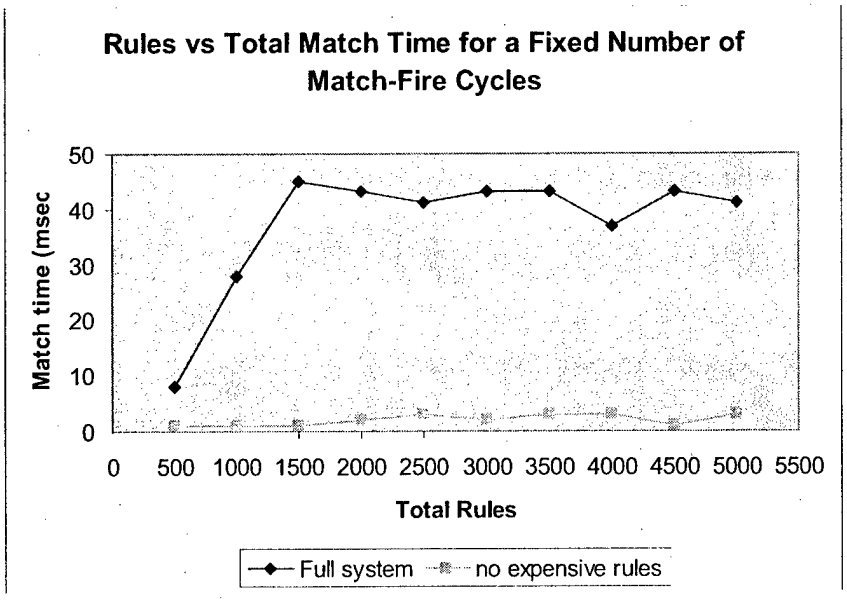

Figure 14: Total match time in milliseconds vs. total rules for a fixed number (10) of match-fire cycles. Numbers in parentheses refer to the number of production firings per match-fire cycle.

constant (consistent with experimental design) and the resulting total match time is also roughly constant. This result shows that match time does not increase (significantly) with the size of the rule set (as Rete predicts), and support the conclusion that the match-fire cycle execution costs are increasing in the previous experiments due to the costs imposed by a serial implementation.

4.2 Hoplite's Results with FPGA-based Rule Processing

Figure 15 shows the timing comparison between the FPGA rule processing and Rete. It's important to note that there are two separate values for the FPGA timing. The first is theoretically derived from the maximum pin delay acquired from Xilinx' implementation step. The second is experimentally determined, using the 20Mz clock from the Virtex-II development board.

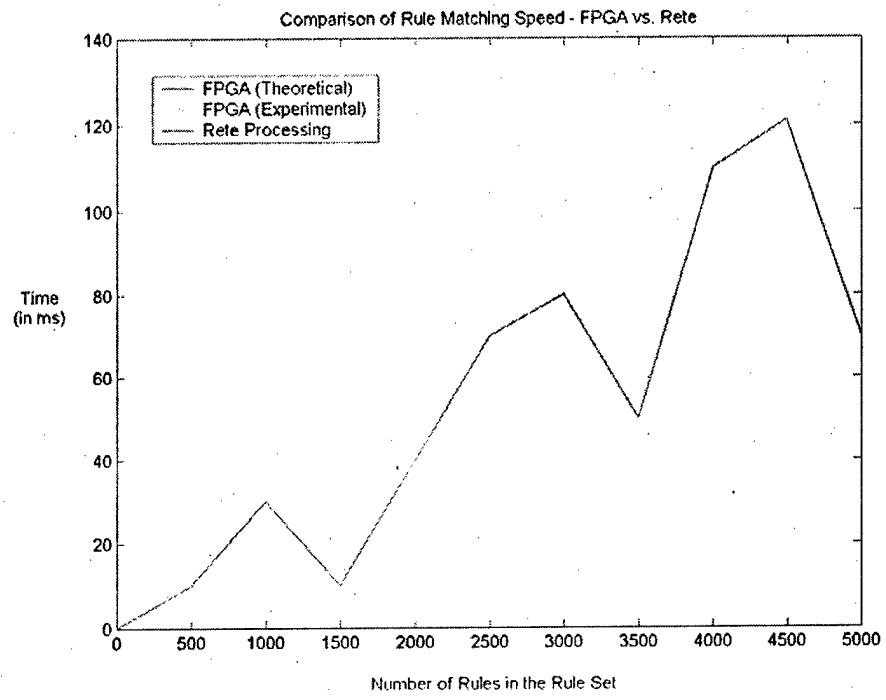

Figure 15: Timing Comparison between Rete and FPGA-based Rule Processing

With the theoretical timing values, the FPGA requires 1.522 ns to process each of the Soar rules. With experimental values, it requires 3.462 ns for each rule. Alternatively, Rete requires 21.4 ms for each cycle. This means that FPGAs provide an average speed-up of 14000x times (theoretical), 6200x (experimental) as compared to the time required by Rete.

This figure isn't particularly helpful due to the magnitude of the difference in results. Table 3 presents the full results for the experiment. This breaks down the rule processing operations for each different set of rules, ranging from 500 to 5000.

Table 3: Summary of productions in the example files.

Number of Rules	FPGA Times (Theoretical)	FPGA Times (Experimental)	Rete Times (Experimental)
500	1.0091	2.5	10000
1000	1.248403	4.45	30000
1500	2.711702	5.45	10000
2000	2.512703	6.95	40000
2500	4.410342	7.95	70000
3000	4.903038	9.45	80000
3500	4.613257	10.45	50000
4000	5.440357	11.95	110000
4500	6.109033	12.95	121000
5000	7.299273	14.45	70000

Although these results may seem to clearly portray FPGA processing as superior, there is more to Soar rule processing than simple combinatorial logic. Soar rules also create pointers from one attribute to another. This associative memory capability is very important to cognitive modeling, but there is no current, accepted means of performing it on a Field Programmable Gate Array.

To make up for this shortcoming, Hoplite Systems proposes to develop a memory-management unit specifically developed for FPGA-based cognitive processing. This is described further in the next section.

5. Conclusions/Recommendations

The results are clear in that FPGAs are superior at processing combinatorial rules. This is because of the parallel, gate-oriented structure of the devices. However, Virtex-II devices have no central memory bus. Therefore, there can be no dynamic memory allocation (DMA) in an FPGA-based circuit. Without DMA, there can be no pointers, and without pointers, there is no way to associate one variable with another. This capability is crucial in predicate memory, and further work in FPGA-based cognitive processing needs this form of memory management.

To make this possible, Hoplite recommends development of a customized memory management unit (MMU) specifically for cognitive operations. The MMU would be implemented with a large-density FPGA and a series of DRAM devices. The execution would be similar to that of a regular CPU's MMU, except for three fundamental differences:

- *multiple connections from the MMU to memory* – this is a distinct improvement over a computer's MMU, which only provides one bus between computation and storage

- *dynamic allocation units for high-volume data transfer* – while a CPU memory bus has a fixed width, the FPGA MMU will set its width according to its processing needs

- *memory can be partitioned for multiple uses* – a CPU MMU treats system memory as a monolithic block, but the FPGA can operate multiple devices in parallel

Our further recommendations for developmental effort center on the three tasks needed to make JBits development a more viable tool for FPGA-based rule matching. First, an application needs to be developed to perform boolean minimization on a set of rules. Second, the results of the minimization need to be implemented column-wise in Configurable Logic Blocks (CLBs). Third, JBits needs a router that can automatically create connections between IOBs, CLBs, and Block RAM.

Developing the boolean minimization application is a straightforward process. Since parsers have already been developed for Soar rules, the main task involves translating Berkeley's open source Espresso algorithm into Java. This algorithm needs to be modified to work with the CLB's LUT structure, but this is the only significant change.

Implementing the set of sum-of-products equations from the minimization tool within CLBs is also straightforward. CLB circuits can be specifically configured for SOP computation, and since all CLBs have the same structure, this configuration can be iterated as often as necessary. The only constraint is that the logic for an individual production should be placed within a single column.

Building an effective JBits router is the most necessary of these recommendations. As shown in Appendix A, it takes more code needed to configure connections than to configure logic. The router must be able to do two things. First, it needs to automatically connect selected points in the FPGA. Second, it must keep track of prior connections to prevent intersections. JBits 3.0 contains a Wire Database API that allows users to create applications that store connections. But there's a great deal of work that needs to be accomplished before this can be used.

6. References

1. Laird, John. *The Soar 8 Tutorial*. University of Michigan. 2003. pp. 112-117.

2. Newell, Allen. *Unified Theories of Cognition*. Harvard University Press. 1990. pp. 194-196.

3. Doorenbos, Robert. *Production Matching for Large Learning Systems*. Carnegie-Mellon. 1995. pp. 7-10.

4. Xilinx. *Virtex-II Platform FPGA Handbook*. Xilinx, Inc. 2001. pp. 38-40.

5. Scarpino, Matthew. *The Hoplite Guide to Run-time Reconfigurable Computing*. Hoplite Systems, LLC. 2004. pp. 62, 66-67.

6. Katz, Randall. *Contemporary Logic Design*. Addison-Wesley. 1991. Section 2.4.

Appendix A: Timing Comparison: Rete/FPGA

The following code shows how the process of Soar/VHDL conversion was performed. Soar Technology provided ten files, each containing a different number of rules. The file with 500 rules was called 500prods.soar, and these files progressed incrementally up to 5000prods.soar. The first file shows the rules contained in 500prods.soar.

To implement rule processing on an FPGA, Hoplite created a Java routine that creates a VHDL file based on each set of rules. This is shown in Convert.java. Essentially, the code functions by searching for variables and attributes within the rules. Then, it determines the minimum number of bits needed for each attribute. With this in place, the conversion application creates a series of concurrent conditional assignments that fire productions based on the incoming Working Memory Elements (WMEs).

The last file in this appendix, `prods500.vhdl`, shows the result of the conversion process. It begins with declarations for input and output signals, and continues with conditional assignments for each production. The outputs provide the assertion and de-assertion of the family members declared in 500prods.soar.

Because the actual code is so long, the `500prods.soar` and `prods500.vhdl` files have been abbreviated.

A.1 500prods.soar

```
sp {create*families
    (state <s> ^superstate nil)
    -->
    (<s> ^family <f*1> + ^family <f*2> + ^family <f*3> + ^family <f*4>
           ^family <f*5> + ^family <f*6> + ^family <f*7> + ^family<f*8>
           ^family <f*9> + ^family <f*10> +)
    (<f*1> ^name washington +)
    (<f*2> ^name adams +)
    (<f*3> ^name jefferson +)
    (<f*4> ^name monroe +)
    (<f*5> ^name jackson +)
    (<f*6> ^name madison +)
    (<f*7> ^name tyler +)
    (<f*8> ^name wilson +)
    (<f*9> ^name lincoln +)
    (<f*10> ^name grant +)
}

sp {create*male-names
    (state <s> ^superstate nil)
    -->
    (<s> ^male-names <b> +)
    (<b> ^name robert + ^name george + ^name frederick + ^name john +
           ^name martin + ^name thomas + ^name abraham + ^name theodore
           ^name franklin + ^name dwight + ^name david + ^name james +
           ^name paul + ^name matthew + ^name mark + ^name luke +
           ^name woodrow + ^name william + ^name howard + ^name daniel
```

```
            ^name stephen + ^name roy + ^name edward + ^name phillip +
            ^name ronald + ^name richard + ^name michael + ^name charles
            ^name gerald + ^name joseph + ^name scott + ^name allen +
            ^name peter + ^name warren + ^name anthony + ^name kevin +
            ^gender male +)
}

sp {create*female-names
    (state <s> ^superstate nil)
    -->
    (<s> ^female-names <b> +)
    (<b> ^name penny + ^name abigal + ^name martha + ^name laura +
           ^name hillary + ^name barbara + ^name nancy + ^name rose +
           ^name betty + ^name patricia + ^name karen + ^name christine
           ^name mary + ^name catherine + ^name joan + ^name evelyn +
           ^name pearl + ^name irma + ^name alma + ^name elsa +
           ^name savannah + ^name kate + ^name elizabeth + ^name
           ^name margaret + ^name alice + ^name teresa + ^name polly +
           ^name sara + ^name norah + ^name anita + ^name edie + ^name
           ^name meryl + ^name jacqueline + ^name mamie + ^gender
            female +)
}

sp {initial*generation
    (state <s> ^superstate nil)
    -->
    (<s> ^generation 0 + ^time-of <t> +)
}

sp {next-generation
    (state <s> ^generation <g>)
    -->
    (<s> ^generation (( <g> 1) +)
}

sp {add-gender-of-child
    (state <s> ^family <f> ^{ << female-names male-names >> <a*1> }
<n>)
    (<f> ^member <m> ^name <fname>)
    (<m> ^name <name>)
    (<n> ^name <name> ^gender <g>)
    -->
    (<m> ^gender <g> +)
}

sp {add-pointer-to-parent
    (state <s> ^family <f>)
    (<f> ^member <c> ^member <p>)
    (<c> ^parent-uid <puid>)
    (<p> ^uid <puid>)
    -->
    (<c> ^parent <p> +)
}

sp {add-children-of-parent
    (state <s> ^family <f>)
    (<f> ^member <c> ^member <p> ^name <fname>)
```

```
    (<c> ^parent <p> ^name <name>)
    (<p> ^name <pname>)
    -->
    (<p> ^child <c> +)
}

sp {time-of-mamie-washington-0-1-2
    (state <s> ^time-of <t> -^generation { > 3 <g*2> }
          ^generation { >= 0 <g*1> })
    -->
    (<t> ^person mamie-washington-0-1-2 +)
}

sp {mamie-washington-0-1-2
    (state <s> ^time-of <t*1> ^family <f>)
    (<t*1> ^person mamie-washington-0-1-2)
    (<f> ^name washington)
    -->
    (<f> ^member <m> +)
    (<m> ^name mamie + ^uid mamie-washington-0-1-2 +)
}

sp {time-of-thomas-washington-1-1-4
    (state <s> ^time-of <t> -^generation { > 4 <g*2> }
          ^generation { >= 1 <g*1> })
    -->
    (<t> ^person thomas-washington-1-1-4 +)
}

sp {thomas-washington-1-1-4
    (state <s> ^time-of <t*1> ^family <f>)
    (<t*1> ^person thomas-washington-1-1-4)
    (<f> ^name washington)
    -->
    (<f> ^member <m> +)
    (<m> ^name thomas + ^parent-uid mamie-washington-0-1-2 +
          ^uid thomas-washington-1-1-4 +)
}

sp {time-of-elsa-washington-1-2-6
    (state <s> ^time-of <t> -^generation { > 4 <g*2> }
          ^generation { >= 1 <g*1> })
    -->
    (<t> ^person elsa-washington-1-2-6 +)
}

sp {elsa-washington-1-2-6
    (state <s> ^time-of <t*1> ^family <f>)
    (<t*1> ^person elsa-washington-1-2-6)
    (<f> ^name washington)
    -->
    (<f> ^member <m> +)
    (<m> ^name elsa + ^parent-uid mamie-washington-0-1-2 +
          ^uid elsa-washington-1-2-6 +)
}

sp {time-of-elsa-washington-1-3-8
```

```
    (state <s> ^time-of <t> -^generation { > 4 <g*2> }
          ^generation { >= 1 <g*1> })
    -->
    (<t> ^person elsa-washington-1-3-8 +)
}

sp {elsa-washington-1-3-8
    (state <s> ^time-of <t*1> ^family <f>)
    (<t*1> ^person elsa-washington-1-3-8)
    (<f> ^name washington)
    -->
    (<f> ^member <m> +)
    (<m> ^name elsa + ^parent-uid mamie-washington-0-1-2 +
          ^uid elsa-washington-1-3-8 +)
}

sp {time-of-alice-washington-1-4-10
    (state <s> ^time-of <t> -^generation { > 4 <g*2> }
          ^generation { >= 1 <g*1> })
    -->
    (<t> ^person alice-washington-1-4-10 +)
}

sp {alice-washington-1-4-10
    (state <s> ^time-of <t*1> ^family <f>)
    (<t*1> ^person alice-washington-1-4-10)
    (<f> ^name washington)
    -->
    (<f> ^member <m> +)
    (<m> ^name alice + ^parent-uid mamie-washington-0-1-2 +
          ^uid alice-washington-1-4-10 +)
}

sp {time-of-karen-washington-1-5-12
    (state <s> ^time-of <t> -^generation { > 4 <g*2> }
          ^generation { >= 1 <g*1> })
    -->
    (<t> ^person karen-washington-1-5-12 +)
}

sp {karen-washington-1-5-12
    (state <s> ^time-of <t*1> ^family <f>)
    (<t*1> ^person karen-washington-1-5-12)
    (<f> ^name washington)
    -->
    (<f> ^member <m> +)
    (<m> ^name karen + ^parent-uid mamie-washington-0-1-2 +
          ^uid karen-washington-1-5-12 +)
}

sp {time-of-joseph-washington-1-6-14
    (state <s> ^time-of <t> -^generation { > 4 <g*2> }
          ^generation { >= 1 <g*1> })
    -->
    (<t> ^person joseph-washington-1-6-14 +)
}
```

```
sp {joseph-washington-1-6-14
    (state <s> ^time-of <t*1> ^family <f>)
    (<t*1> ^person joseph-washington-1-6-14)
    (<f> ^name washington)
    -->
    (<f> ^member <m> +)
    (<m> ^name joseph + ^parent-uid mamie-washington-0-1-2 +
          ^uid joseph-washington-1-6-14 +)
}

sp {time-of-edie-washington-1-7-16
    (state <s> ^time-of <t> -^generation { > 4 <g*2> }
          ^generation { >= 1 <g*1> })
    -->
    (<t> ^person edie-washington-1-7-16 +)
}

sp {edie-washington-1-7-16
    (state <s> ^time-of <t*1> ^family <f>)
    (<t*1> ^person edie-washington-1-7-16)
    (<f> ^name washington)
    -->
    (<f> ^member <m> +)
    (<m> ^name edie + ^parent-uid mamie-washington-0-1-2 +
          ^uid edie-washington-1-7-16 +)
}

sp {time-of-irma-washington-1-8-18
    (state <s> ^time-of <t> -^generation { > 4 <g*2> }
          ^generation { >= 1 <g*1> })
    -->
    (<t> ^person irma-washington-1-8-18 +)
}

sp {irma-washington-1-8-18
    (state <s> ^time-of <t*1> ^family <f>)
    (<t*1> ^person irma-washington-1-8-18)
    (<f> ^name washington)
    -->
    (<f> ^member <m> +)
    (<m> ^name irma + ^parent-uid mamie-washington-0-1-2 +
          ^uid irma-washington-1-8-18 +)
}

sp {time-of-phillip-washington-1-9-20
    (state <s> ^time-of <t> -^generation { > 4 <g*2> }
          ^generation { >= 1 <g*1> })
    -->
    (<t> ^person phillip-washington-1-9-20 +)
}

sp {phillip-washington-1-9-20
    (state <s> ^time-of <t*1> ^family <f>)
    (<t*1> ^person phillip-washington-1-9-20)
    (<f> ^name washington)
    -->
    (<f> ^member <m> +)
```

```
    (<m> ^name phillip + ^parent-uid mamie-washington-0-1-2 +
          ^uid phillip-washington-1-9-20 +)
}
```

A.2 Convert.java

```
package converter;

import java.io.*;

public class Convert {

  static String filename = "500";
  boolean male=false;

  static String[] families = {"washington", "adams", "jefferson",
    "monroe", "jackson", "madison", "tyler", "wilson", "lincoln",
    "grant"};

  static String[] names = {"penny", "abigal", "martha", "laura",
    "hillary", "barbara", "nancy", "rose", "betty", "patricia",
    "karen", "christine", "mary", "catherine", "joan", "evelyn",
    "pearl", "irma", "alma", "elsa", "savannah", "kate",
    "elizabeth", "victoria", "margaret", "alice", "teresa", "polly",
    "sara", "norah", "anita", "edie", "mia", "meryl", "jacqueline",
    "mamie", "robert", "george", "frederick", "john", "martin",
    "thomas", "abraham", "theodore", "franklin", "dwight", "david",
    "james", "paul", "matthew", "mark", "luke", "woodrow", "william",
    "howard", "daniel",
    "stephen", "roy", "edward", "phillip", "ronald", "richard",
    "michael", "charles", "gerald", "joseph",
    "scott", "allen", "peter", "warren", "anthony", "kevin"};

  public static void main(String[] args) throws IOException {

    String num, line, fname, lname, gen, step, prodval;
    int prodint, pos, dash1, dash2, dash3, dash4, para, found, maxgens,
      maxsteps, gennum, stepnum, genval;
    int[] count = new int[10];
    String[] born = new String[10];
    String[] dead = new String[10];
    int prod, maxprods, namenum, familynum;

    File f = new File(filename+"prods.soar");
    BufferedReader br = new BufferedReader(new FileReader(f));
    line = br.readLine();
    br.mark((int) f.length());
    for (int i=0; i<10; i++)
      count[i] = 0;
    maxgens = 0; maxsteps = 0; maxprods = 0;
    while (line != null) {
      pos = line.indexOf("time-of-");
      if (pos != -1) {
        dash1 = line.indexOf("-", pos+8);
        dash2 = line.indexOf("-", dash1+1);
        dash3 = line.indexOf("-", dash2+1);
```

```
      dash4 = line.indexOf("-", dash3+1);
      gen = line.substring(dash2+1, dash3);
      gennum = Integer.decode(gen).intValue();
      if (gennum > maxgens)
        maxgens = gennum;
      step = line.substring(dash3+1, dash4);
      stepnum = Integer.decode(step).intValue();
      if (stepnum > maxsteps)
        maxsteps = stepnum;
      prod = Integer.decode(line.substring(dash4+1)).intValue();
      if (prod > maxprods)
        maxprods = prod;
    }
    line = br.readLine();
  }

  br.reset();
  System.out.println(br.readLine());

  gennum = (int) Math.ceil(Math.log(maxgens+4)/Math.log(2));
  stepnum = (int) Math.ceil(Math.log(maxsteps+1)/Math.log(2));
  prod = (int) Math.ceil(Math.log(maxprods)/Math.log(2));
  namenum = (int) Math.ceil(Math.log(names.length)/Math.log(2));
  familynum = (int) Math.ceil(Math.log(families.length)/Math.log(2));

  System.out.println("Name bits: "+namenum);
  System.out.println("Family bits: "+familynum);
  System.out.println("Max generations: "+maxgens+" Generation bits:
    "+gennum);
  System.out.println("Step bits: "+stepnum);
  System.out.println("Production bits: "+prod);

  BufferedWriter bw = new BufferedWriter(new
    FileWriter("prods"+filename+"_main.vd"));
  bw.write("library IEEE;\n");
  bw.write("use IEEE.STD_LOGIC_1164.ALL;\n");
  bw.write("use IEEE.STD_LOGIC_ARITH.ALL;\n");
  bw.write("use IEEE.STD_LOGIC_UNSIGNED.ALL;\n\n");
  bw.write("entity prods"+filename+"_main is\n");
  bw.write("    Port ( clock : in std_logic;\n");
  bw.write("           reset : in std_logic;\n");
  for (int i=0; i<families.length-1; i++) {
    bw.write("           "+families[i].toUpperCase()+"_asserted : out
      std_logic_vector("+(namenum+familynum+gennum+stepnum+prod-1)+"
      downto 0);\n");
    bw.write("           "+families[i].toUpperCase()+"_deassert : out
      std_logic_vector("+(namenum+familynum+gennum+stepnum+prod-1)+"
      downto 0);\n");
  }
  bw.write("           "+families[families.length-1].toUpperCase()+
    "_asserted : out std_logic_vector("+(namenum+familynum+gennum+
    stepnum+prod-1)+" downto 0);\n");
  bw.write("           "+families[families.length-1].toUpperCase()+
    "_deassert : out std_logic_vector("+(namenum+familynum+gennum+
    stepnum+prod-1)+" downto 0));\n");
  bw.write("end prods"+filename+"_main;\n\n");
  bw.write("architecture Behavioral of prods"+filename+"_main
```

```
  is\n\n");
bw.write("    signal sig_clock: std_logic;\n");
bw.write("    signal sig_intime:
  std_logic_vector("+(gennum+stepnum-1)+" downto 0);\n");
bw.write("\n    component genstep"+filename+" is\n");
bw.write("        port ( reset : in std_logic;\n");
bw.write("               clock : in std_logic;\n");
bw.write("               genstep : out
  std_logic_vector("+(gennum+stepnum-1)+" downto 0));\n");
bw.write("    end component;\n\n");
bw.write("    component prods"+filename+" is\n");
bw.write("        port ( intime : in
  std_logic_vector("+(gennum+stepnum-1)+" downto 0);\n");

for (int i=0; i<families.length-1; i++) {
  bw.write("              "+families[i].toUpperCase()+"_asserted :
  out std_logic_vector("+(namenum+familynum+gennum+stepnum+prod-1)
    +" downto 0);\n");
  bw.write("              "+families[i].toUpperCase()+"_deassert :
    out std_logic_vector("+(namenum+familynum+gennum+stepnum+prod-
    1)+" downto 0);\n");
}

bw.write("              "+families[families.length-1]
  .toUpperCase()+"_asserted : out std_logic_vector("+
  (namenum+familynum+gennum+stepnum+prod-1)+" downto 0);\n");
bw.write("              "+families[families.length-1].
  toUpperCase()+"_deassert : out std_logic_vector("
  +(namenum+familynum+gennum+stepnum+prod-1)+"
  downto 0));\n");
bw.write("    end component;\n\n");
bw.write("begin\n\n");
bw.write("    process(clock)\n");
bw.write("    begin\n");
bw.write("            if rising_edge(clock) then\n");
bw.write("                  sig_clock <= not sig_clock;\n");
bw.write("            end if;\n");
bw.write("    end process;\n\n");
bw.write("    pc: genstep"+filename+" \n");
bw.write("        port map(reset => reset, clock => sig_clock,
  genstep => sig_intime);\n\n");
bw.write("    pro: prods"+filename+" \n");
bw.write("        port map(\n");
bw.write("           intime => sig_intime,\n");

for (int i=0; i<families.length-1; i++) {
  bw.write("          "+families[i].toUpperCase()+"_asserted
    => "+families[i].toUpperCase()+"_asserted,\n");
  bw.write("          "+families[i].toUpperCase()+"_deassert
    => "+families[i].toUpperCase()+"_deassert,\n");
}
bw.write("          "+families[families.length-
  1].toUpperCase()+"_asserted => "+families[families.length-
  1].toUpperCase()+"_asserted,\n");
bw.write("          "+families[families.length-
  1].toUpperCase()+"_deassert => "+families[families.length-
  1].toUpperCase()+"_deassert);\n");
```

```
    bw.write("end Behavioral;\n");
    br.close();
    bw.close();
  }
}
```

A.3 prods500.vhdl

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity prods500 is
    Port (
          intime: in std_logic_vector(6 downto 0);
          WASHINGTON_asserted : out std_logic_vector(26 downto 0);
          WASHINGTON_deassert : out std_logic_vector(26 downto 0);
          ADAMS_asserted : out std_logic_vector(26 downto 0);
          ADAMS_deassert : out std_logic_vector(26 downto 0);
          JEFFERSON_asserted : out std_logic_vector(26 downto 0);
          JEFFERSON_deassert : out std_logic_vector(26 downto 0);
          MONROE_asserted : out std_logic_vector(26 downto 0);
          MONROE_deassert : out std_logic_vector(26 downto 0);
          JACKSON_asserted : out std_logic_vector(26 downto 0);
          JACKSON_deassert : out std_logic_vector(26 downto 0);
          MADISON_asserted : out std_logic_vector(26 downto 0);
          MADISON_deassert : out std_logic_vector(26 downto 0);
          TYLER_asserted : out std_logic_vector(26 downto 0);
          TYLER_deassert : out std_logic_vector(26 downto 0);
          WILSON_asserted : out std_logic_vector(26 downto 0);
          WILSON_deassert : out std_logic_vector(26 downto 0);
          LINCOLN_asserted : out std_logic_vector(26 downto 0);
          LINCOLN_deassert : out std_logic_vector(26 downto 0);
          GRANT_asserted : out std_logic_vector(26 downto 0);
          GRANT_deassert : out std_logic_vector(26 downto 0));
end prods500;

architecture Behavioral of prods500 is

    constant GEN0: std_logic_vector(2 downto 0) := "000";
    constant GEN1: std_logic_vector(2 downto 0) := "001";
    constant GEN2: std_logic_vector(2 downto 0) := "010";
    constant GEN3: std_logic_vector(2 downto 0) := "011";
    constant GEN4: std_logic_vector(2 downto 0) := "100";
    constant GEN5: std_logic_vector(2 downto 0) := "101";

    constant STEP1: std_logic_vector(3 downto 0) := "0000";
    constant STEP2: std_logic_vector(3 downto 0) := "0001";
    constant STEP3: std_logic_vector(3 downto 0) := "0010";
    constant STEP4: std_logic_vector(3 downto 0) := "0011";
    constant STEP5: std_logic_vector(3 downto 0) := "0100";
    constant STEP6: std_logic_vector(3 downto 0) := "0101";
    constant STEP7: std_logic_vector(3 downto 0) := "0110";
    constant STEP8: std_logic_vector(3 downto 0) := "0111";
    constant STEP9: std_logic_vector(3 downto 0) := "1000";
```

```
constant STEP10: std_logic_vector(3 downto 0) := "1001";
constant STEP11: std_logic_vector(3 downto 0) := "1010";
constant STEP12: std_logic_vector(3 downto 0) := "1011";
constant STEP13: std_logic_vector(3 downto 0) := "1100";
constant STEP14: std_logic_vector(3 downto 0) := "1101";

constant WASHINGTON: std_logic_vector(3 downto 0) := "0001";
constant ADAMS: std_logic_vector(3 downto 0) := "0010";
constant JEFFERSON: std_logic_vector(3 downto 0) := "0011";
constant MONROE: std_logic_vector(3 downto 0) := "0100";
constant JACKSON: std_logic_vector(3 downto 0) := "0101";
constant MADISON: std_logic_vector(3 downto 0) := "0110";
constant TYLER: std_logic_vector(3 downto 0) := "0111";
constant WILSON: std_logic_vector(3 downto 0) := "1000";
constant LINCOLN: std_logic_vector(3 downto 0) := "1001";
constant GRANT: std_logic_vector(3 downto 0) := "1010";

constant PENNY: std_logic_vector(6 downto 0) := "0000001";
constant ABIGAL: std_logic_vector(6 downto 0) := "0000010";
constant MARTHA: std_logic_vector(6 downto 0) := "0000011";
constant LAURA: std_logic_vector(6 downto 0) := "0000100";
constant HILLARY: std_logic_vector(6 downto 0) := "0000101";
constant BARBARA: std_logic_vector(6 downto 0) := "0000110";
constant NANCY: std_logic_vector(6 downto 0) := "0000111";
constant ROSE: std_logic_vector(6 downto 0) := "0001000";
constant BETTY: std_logic_vector(6 downto 0) := "0001001";
constant PATRICIA: std_logic_vector(6 downto 0) := "0001010";
constant KAREN: std_logic_vector(6 downto 0) := "0001011";
constant CHRISTINE: std_logic_vector(6 downto 0) := "0001100";
constant MARY: std_logic_vector(6 downto 0) := "0001101";
constant CATHERINE: std_logic_vector(6 downto 0) := "0001110";
constant JOAN: std_logic_vector(6 downto 0) := "0001111";
constant EVELYN: std_logic_vector(6 downto 0) := "0010000";
constant PEARL: std_logic_vector(6 downto 0) := "0010001";
constant IRMA: std_logic_vector(6 downto 0) := "0010010";
constant ALMA: std_logic_vector(6 downto 0) := "0010011";
constant ELSA: std_logic_vector(6 downto 0) := "0010100";
constant SAVANNAH: std_logic_vector(6 downto 0) := "0010101";
constant KATE: std_logic_vector(6 downto 0) := "0010110";
constant ELIZABETH: std_logic_vector(6 downto 0) := "0010111";
constant VICTORIA: std_logic_vector(6 downto 0) := "0011000";
constant MARGARET: std_logic_vector(6 downto 0) := "0011001";
constant ALICE: std_logic_vector(6 downto 0) := "0011010";
constant TERESA: std_logic_vector(6 downto 0) := "0011011";
constant POLLY: std_logic_vector(6 downto 0) := "0011100";
constant SARA: std_logic_vector(6 downto 0) := "0011101";
constant NORAH: std_logic_vector(6 downto 0) := "0011110";
constant ANITA: std_logic_vector(6 downto 0) := "0011111";
constant EDIE: std_logic_vector(6 downto 0) := "0100000";
constant MIA: std_logic_vector(6 downto 0) := "0100001";
constant MERYL: std_logic_vector(6 downto 0) := "0100010";
constant JACQUELINE: std_logic_vector(6 downto 0) := "0100011";
constant MAMIE: std_logic_vector(6 downto 0) := "0100100";
constant ROBERT: std_logic_vector(6 downto 0) := "0100101";
constant GEORGE: std_logic_vector(6 downto 0) := "0100110";
constant FREDERICK: std_logic_vector(6 downto 0) := "0100111";
constant JOHN: std_logic_vector(6 downto 0) := "0101000";
```

```
    constant MARTIN: std_logic_vector(6 downto 0) := "0101001";
    constant THOMAS: std_logic_vector(6 downto 0) := "0101010";
    constant ABRAHAM: std_logic_vector(6 downto 0) := "0101011";
    constant THEODORE: std_logic_vector(6 downto 0) := "0101100";
    constant FRANKLIN: std_logic_vector(6 downto 0) := "0101101";
    constant DWIGHT: std_logic_vector(6 downto 0) := "0101110";
    constant DAVID: std_logic_vector(6 downto 0) := "0101111";
    constant JAMES: std_logic_vector(6 downto 0) := "0110000";
    constant PAUL: std_logic_vector(6 downto 0) := "0110001";
    constant MATTHEW: std_logic_vector(6 downto 0) := "0110010";
    constant MARK: std_logic_vector(6 downto 0) := "0110011";
    constant LUKE: std_logic_vector(6 downto 0) := "0110100";
    constant WOODROW: std_logic_vector(6 downto 0) := "0110101";
    constant WILLIAM: std_logic_vector(6 downto 0) := "0110110";
    constant HOWARD: std_logic_vector(6 downto 0) := "0110111";
    constant DANIEL: std_logic_vector(6 downto 0) := "0111000";
    constant STEPHEN: std_logic_vector(6 downto 0) := "0111001";
    constant ROY: std_logic_vector(6 downto 0) := "0111010";
    constant EDWARD: std_logic_vector(6 downto 0) := "0111011";
    constant PHILLIP: std_logic_vector(6 downto 0) := "0111100";
    constant RONALD: std_logic_vector(6 downto 0) := "0111101";
    constant RICHARD: std_logic_vector(6 downto 0) := "0111110";
    constant MICHAEL: std_logic_vector(6 downto 0) := "0111111";
    constant CHARLES: std_logic_vector(6 downto 0) := "1000000";
    constant GERALD: std_logic_vector(6 downto 0) := "1000001";
    constant JOSEPH: std_logic_vector(6 downto 0) := "1000010";
    constant SCOTT: std_logic_vector(6 downto 0) := "1000011";
    constant ALLEN: std_logic_vector(6 downto 0) := "1000100";
    constant PETER: std_logic_vector(6 downto 0) := "1000101";
    constant WARREN: std_logic_vector(6 downto 0) := "1000110";
    constant ANTHONY: std_logic_vector(6 downto 0) := "1000111";
    constant KEVIN: std_logic_vector(6 downto 0) := "1001000";

begin
  with intime select
    WASHINGTON_asserted <=

    MAMIE & WASHINGTON & GEN0 & STEP1 & "000000010" when GEN0 & STEP1,
    THOMAS & WASHINGTON & GEN1 & STEP1 & "000000100" when GEN1 & STEP1,
    ELSA & WASHINGTON & GEN1 & STEP2 & "000000110" when GEN1 & STEP2,
    ELSA & WASHINGTON & GEN1 & STEP3 & "000001000" when GEN1 & STEP3,
    ALICE & WASHINGTON & GEN1 & STEP4 & "000001010" when GEN1 & STEP4,
    KAREN & WASHINGTON & GEN1 & STEP5 & "000001100" when GEN1 & STEP5,
    JOSEPH & WASHINGTON & GEN1 & STEP6 & "000001110" when GEN1 & STEP6,
    EDIE & WASHINGTON & GEN1 & STEP7 & "000010000" when GEN1 & STEP7,
    IRMA & WASHINGTON & GEN1 & STEP8 & "000010010" when GEN1 & STEP8,
    PHILLIP & WASHINGTON & GEN1 & STEP9 & "000010100" when GEN1 &
      STEP9,
    PETER & WASHINGTON & GEN1 & STEP10 & "000010110" when GEN1 &
      STEP10,
    PAUL & WASHINGTON & GEN2 & STEP1 & "000011000" when GEN2 & STEP1,
    ELSA & WASHINGTON & GEN2 & STEP2 & "000011010" when GEN2 & STEP2,
    HOWARD & WASHINGTON & GEN2 & STEP3 & "000011100" when GEN2 & STEP3,
    BARBARA & WASHINGTON & GEN2 & STEP4 & "000011110" when GEN2 &
      STEP4,
    HILLARY & WASHINGTON & GEN2 & STEP5 & "000100000" when GEN2 &
      STEP5,
```

```
PHILLIP & WASHINGTON & GEN2 & STEP6 & "000100010" when GEN2 &
  STEP6,
PEARL & WASHINGTON & GEN2 & STEP7 & "000100100" when GEN2 & STEP7,
THEODORE & WASHINGTON & GEN2 & STEP8 & "000100110" when GEN2 &
  STEP8,
RICHARD & WASHINGTON & GEN2 & STEP9 & "000101000" when GEN2 &
  STEP9,
SCOTT & WASHINGTON & GEN2 & STEP10 & "000101010" when GEN2 &
  STEP10,
RICHARD & WASHINGTON & GEN2 & STEP11 & "000101100" when GEN2 &
  STEP11,
BARBARA & WASHINGTON & GEN2 & STEP12 & "000101110" when GEN2 &
  STEP12,
ROSE & WASHINGTON & GEN2 & STEP13 & "000110000" when GEN2 & STEP13,
PATRICIA & WASHINGTON & GEN2 & STEP14 & "000110010" when GEN2 &
  STEP14,
"00000000000000000000000000" when others;
```

Appendix B: RAISE Circuitry Code (Java/JBits)

The following code presents the different classes that configure the Waterjug computation circuit inside a XC2V1000 FPGA. Most of the classes configure a specific tile type (CLB, IOB, Term) in multiple places throughout the device. The `RAISE_main` class performs configuration by invoking the `configure()` method in each of the other classes. The end result is the RAISE.bit file that can be transferred directly to the FPGA.

B.1 RAISE_main.java

```
import com.xilinx.JBits.Virtex2.*;
import com.xilinx.JBits.ArchIndependent.ConfigurationException;

public class RAISE_main {

  public static void main(String[] args)
    throws ConfigurationException {

    Device device = Device.getDevice("XC2V1000");  // Creates a new
    JBits jbits = new JBits(device);               // JBits object

    try {
      jbits.read("in.bit");
    } catch (ConfigurationException ce) {
      System.out.println("Problem reading the bitstream.\n" + ce);
    }

    RAISE_IOIs.configure(jbits);     // Configures the I/O Tiles
    RAISE_clks.configure(jbits);     // Configures the Clock Tiles
    RAISE_CLBs.configure(jbits);     // Configures the CLBs
    RAISE_Bterms.configure(jbits);   // Configures the bottom Term Tiles
    RAISE_Lterms.configure(jbits);   // Configures the left Term Tiles
    RAISE_Tterms.configure(jbits);   // Configures the top Term Tiles
```

```
    try {
      jbits.write("RAISE.bit", JBits.FULL);  // Creates the final
    } catch (ConfigurationException ce) {    // bitstream
      System.out.println("Problem writing the bitstream.\n" + ce);
    }
  }
}
```

B.2 RAISE_IOIs.java

```
import com.xilinx.JBits.ArchIndependent.ConfigurationException;
import com.xilinx.JBits.Virtex2.*;
import com.xilinx.JBits.Virtex2.Bits.Logic.IOIs.*;
import com.xilinx.JBits.Virtex2.Bits.Wires.IOIs.*;

public class RAISE_IOIs {
  public static void configure(JBits jbits) {

    try {
      // Configuration for the IOIs at (33, 1)
      jbits.setTileBits(33, 1, S2BEG4.S2BEG4, S2BEG4.W2MID4);

      // Configuration for the IOIs at (32, 1)
      jbits.setTileBits(32, 1, OMUX.CONFIG[1], OMUX.O1);
      jbits.setTileBits(32, 1, T1INV.CONFIG[1], T1INV.T1_B);
      jbits.setTileBits(32, 1, TMUX.CONFIG[1], TMUX.T1);
      jbits.setTileBits(32, 1, O1_B1.O1_B1, O1_B1.S2MID4);

      // Configuration for the IOIs at (49, 10)
      jbits.setTileBits(49, 10, OMUX.CONFIG[2], OMUX.O1);
      jbits.setTileBits(49, 10, T1INV.CONFIG[2], T1INV.T1_B);
      jbits.setTileBits(49, 10, TMUX.CONFIG[2], TMUX.T1);
      jbits.setTileBits(49, 10, E6BEG0.E6BEG0, E6BEG0.I3);
      jbits.setTileBits(49, 10, FAN_BX1.FAN_BX1, FAN_BX1.W2MID7);
      jbits.setTileBits(49, 10, O1_B2.O1_B2, O1_B2.FAN_BX1);

      // Configuration for the IOIs at (49, 11)
      jbits.setTileBits(49, 11, W2BEG7.W2BEG7, W2BEG7.N2END6);

    } catch (ConfigurationException ce) {
      System.out.println("Problem modifying the bitstream." + ce);
    }
  }
}
```

B.3 RAISE_clks.java

```
import com.xilinx.JBits.ArchIndependent.ConfigurationException;
import com.xilinx.JBits.Virtex2.*;
import com.xilinx.JBits.Virtex2.Bits.Logic.Gclkh.*;
import com.xilinx.JBits.Virtex2.Bits.Wires.Gclkh.*;
import com.xilinx.JBits.Virtex2.Bits.Wires.Clkvul.*;
```

```
import com.xilinx.JBits.Virtex2.Bits.Wires.Clkvu2.*;
import com.xilinx.JBits.Virtex2.Bits.Wires.Clkvd1.*;
import com.xilinx.JBits.Virtex2.Bits.Wires.Clkvd2.*;

public class RAISE_clks {
  public static void configure(JBits jbits) {

    // This class configures the horizontal (Gclkh) and vertical
    // (Gclkv) tiles within the FPGA.

    try {
      // Configuration for the Gclkh at (44, 2)
      jbits.setTileBits(44, 2, GCLK_UP1.GCLK_UP1, GCLK_UP1.GCLK_B1);

      // Configuration for the Gclkh at (34, 2)
      jbits.setTileBits(34, 2, GCLK_UP1.GCLK_UP1, GCLK_UP1.GCLK_B1);

      // Configuration for the Gclkh at (16, 2)
      jbits.setTileBits(16, 2, GCLK_DN4.GCLK_DN4, GCLK_DN4.GCLK_B4);

      // Configuration for the Gclkh at (6, 2)
      jbits.setTileBits(6, 2, GCLK_DN4.GCLK_DN4, GCLK_DN4.GCLK_B4);

      // Configuration for the Gclkh at (44, 3)
      jbits.setTileBits(44, 3, GCLK_UP1.GCLK_UP1, GCLK_UP1.GCLK_B1);

      // Configuration for the Gclkh at (34, 3)
      jbits.setTileBits(34, 3, GCLK_UP1.GCLK_UP1, GCLK_UP1.GCLK_B1);

      // Configuration for the Gclkh at (44, 6)
      jbits.setTileBits(44, 6, GCLK_UP1.GCLK_UP1, GCLK_UP1.GCLK_B1);

      // Configuration for the Gclkh at (34, 6)
      jbits.setTileBits(34, 6, GCLK_UP1.GCLK_UP1, GCLK_UP1.GCLK_B1);

      // Configuration for the Gclkh at (44, 7)
      jbits.setTileBits(44, 7, GCLK_UP1.GCLK_UP1, GCLK_UP1.GCLK_B1);

      // Configuration for the Gclkh at (34, 7)
      jbits.setTileBits(34, 7, GCLK_UP1.GCLK_UP1, GCLK_UP1.GCLK_B1);

      // Configuration for the Gclkh at (44, 8)
      jbits.setTileBits(44, 8, GCLK_UP1.GCLK_UP1, GCLK_UP1.GCLK_B1);

      // Configuration for the Gclkh at (34, 8)
      jbits.setTileBits(34, 8, GCLK_UP1.GCLK_UP1, GCLK_UP1.GCLK_B1);

      // Configuration for the Gclkh at (44, 9)
      jbits.setTileBits(44, 9, GCLK_UP1.GCLK_UP1, GCLK_UP1.GCLK_B1);

      // Configuration for the Gclkh at (34, 9)
      jbits.setTileBits(34, 9, GCLK_UP1.GCLK_UP1, GCLK_UP1.GCLK_B1);

      // Configuration for the Gclkh at (44, 10)
      jbits.setTileBits(44, 10, DOWN1MUX.CONFIG, DOWN1MUX.OFF);
      jbits.setTileBits(44, 10, GCLK_DN5.GCLK_DN5, GCLK_DN5.GCLK_B5);
      jbits.setTileBits(44, 10, GCLK_UP1.GCLK_UP1, GCLK_UP1.GCLK_B1);
```

```
// Configuration for the Gclkh at (34, 10)
jbits.setTileBits(34, 10, DOWN1MUX.CONFIG, DOWN1MUX.OFF);
jbits.setTileBits(34, 10, GCLK_DN5.GCLK_DN5, GCLK_DN5.GCLK_B5);
jbits.setTileBits(34, 10, GCLK_UP1.GCLK_UP1, GCLK_UP1.GCLK_B1);

// Configuration for the Gclkh at (44, 11)
jbits.setTileBits(44, 11, GCLK_UP1.GCLK_UP1, GCLK_UP1.GCLK_B1);

// Configuration for the Gclkh at (34, 11)
jbits.setTileBits(34, 11, GCLK_UP1.GCLK_UP1, GCLK_UP1.GCLK_B1);

// Configuration for the Gclkh at (44, 12)
jbits.setTileBits(44, 12, GCLK_UP1.GCLK_UP1, GCLK_UP1.GCLK_B1);

// Configuration for the Gclkh at (34, 12)
jbits.setTileBits(34, 12, GCLK_UP1.GCLK_UP1, GCLK_UP1.GCLK_B1);

// Configuration for the Gclkh at (44, 13)
jbits.setTileBits(44, 13, GCLK_UP1.GCLK_UP1, GCLK_UP1.GCLK_B1);

// Configuration for the Gclkh at (34, 13)
jbits.setTileBits(34, 13, GCLK_UP1.GCLK_UP1, GCLK_UP1.GCLK_B1);

// Configuration for the Gclkh at (44, 14)
jbits.setTileBits(44, 14, GCLK_UP1.GCLK_UP1, GCLK_UP1.GCLK_B1);

// Configuration for the Gclkh at (34, 14)
jbits.setTileBits(34, 14, GCLK_UP1.GCLK_UP1, GCLK_UP1.GCLK_B1);

// Configuration for the Gclkh at (44, 15)
jbits.setTileBits(44, 15, GCLK_UP1.GCLK_UP1, GCLK_UP1.GCLK_B1);

// Configuration for the Gclkh at (34, 15)
jbits.setTileBits(34, 15, GCLK_UP1.GCLK_UP1, GCLK_UP1.GCLK_B1);

// Configuration for the Gclkh at (44, 16)
jbits.setTileBits(44, 16, GCLK_UP1.GCLK_UP1, GCLK_UP1.GCLK_B1);

// Configuration for the Gclkh at (34, 16)
jbits.setTileBits(34, 16, GCLK_UP1.GCLK_UP1, GCLK_UP1.GCLK_B1);

// Configuration for the Gclkh at (44, 17)
jbits.setTileBits(44, 17, GCLK_UP1.GCLK_UP1, GCLK_UP1.GCLK_B1);

// Configuration for the Gclkh at (34, 17)
jbits.setTileBits(34, 17, GCLK_UP1.GCLK_UP1, GCLK_UP1.GCLK_B1);

// Configuration for the Gclkh at (44, 20)
jbits.setTileBits(44, 20, GCLK_UP1.GCLK_UP1, GCLK_UP1.GCLK_B1);

// Configuration for the Gclkh at (34, 20)
jbits.setTileBits(34, 20, GCLK_UP1.GCLK_UP1, GCLK_UP1.GCLK_B1);

// Configuration for the Gclkh at (44, 21)
jbits.setTileBits(44, 21, GCLK_DN1.GCLK_DN1, GCLK_DN1.GCLK_B1);
jbits.setTileBits(44, 21, GCLK_DN2.GCLK_DN2, GCLK_DN2.GCLK_B2);
```

```
      jbits.setTileBits(44, 21, GCLK_UP0.GCLK_UP0, GCLK_UP0.GCLK_B0);

      // Configuration for the Gclkh at (34, 21)
      jbits.setTileBits(34, 21, GCLK_DN1.GCLK_DN1, GCLK_DN1.GCLK_B1);
      jbits.setTileBits(34, 21, GCLK_DN2.GCLK_DN2, GCLK_DN2.GCLK_B2);
      jbits.setTileBits(34, 21, GCLK_UP0.GCLK_UP0, GCLK_UP0.GCLK_B0);

      // Configuration for the Clkvu2 at (8, 22)
      jbits.setTileBits(8, 22, GCLKC_GCLKL4.GCLKC_GCLKL4,
        GCLKC_GCLKL4.GCLKC_GCLKT4);

      // Configuration for the Gclkh at (44, 23)
      jbits.setTileBits(44, 23, GCLK_UP1.GCLK_UP1, GCLK_UP1.GCLK_B1);

      // Configuration for the Gclkh at (34, 23)
      jbits.setTileBits(34, 23, GCLK_UP1.GCLK_UP1, GCLK_UP1.GCLK_B1);

      // Configuration for the Gclkh at (44, 24)
      jbits.setTileBits(44, 24, GCLK_UP1.GCLK_UP1, GCLK_UP1.GCLK_B1);

    } catch (ConfigurationException ce) {
      System.out.println("Problem modifying the bitstream." + ce);
    }
  }
}
```

B.4 RAISE_CLBs.java

```
// This class configures the Configurable Logic Blocks inside an FPGA.
// The main difficulty isn't the CLBs themselves, but routing
// connections between them and other resources on the device.

import com.xilinx.JBits.ArchIndependent.ConfigurationException;
import com.xilinx.JBits.Virtex2.*;
import com.xilinx.JBits.Virtex2.Bits.Logic.Center.*;
import com.xilinx.JBits.Virtex2.Bits.Wires.Center.*;

public class RAISE_CLBs {
  public static void configure(JBits jbits) {

    try {
      // Configuration for the Center tile at (33, 2)
      jbits.setTileBits(33, 2, W2BEG4.W2BEG4, W2BEG4.W6END4);

      // Configuration for the Center tile at (5, 2)
      jbits.setTileBits(5, 2, LH6.LH6, LH6.OMUX_N15);

      // Configuration for the Center tile at (4, 2)
      jbits.setTileBits(4, 2, BYINV.CONFIG[1], BYINV.BY_B);
      jbits.setTileBits(4, 2, BYINV.CONFIG[2], BYINV.BY_B);
      jbits.setTileBits(4, 2, CEINV.CONFIG[0], CEINV.CE);
      jbits.setTileBits(4, 2, CEINV.CONFIG[1], CEINV.CE);
      jbits.setTileBits(4, 2, CEINV.CONFIG[2], CEINV.CE);
      jbits.setTileBits(4, 2, CY0F.CONFIG[0], CY0F.ZERO);
      jbits.setTileBits(4, 2, CYINIT.CONFIG[0], CYINIT.CIN);
```

```
      jbits.setTileBits(4, 2, CYSELF.CONFIG[0], CYSELF.F);
      jbits.setTileBits(4, 2, DXMUX.CONFIG[0], DXMUX.DX);
      jbits.setTileBits(4, 2, DYMUX.CONFIG[0], DYMUX.DY);
      jbits.setTileBits(4, 2, FFX_INIT_ATTR.CONFIG[0],
FFX_INIT_ATTR.INIT0);
      jbits.setTileBits(4, 2, FFX_SR_ATTR.CONFIG[0],
FFX_SR_ATTR.SRLOW);
      jbits.setTileBits(4, 2, FFX_SR_ATTR.CONFIG[1],
FFX_SR_ATTR.SRLOW);
      jbits.setTileBits(4, 2, FFY_INIT_ATTR.CONFIG[0],
FFY_INIT_ATTR.INIT0);
      jbits.setTileBits(4, 2, FFY_INIT_ATTR.CONFIG[1],
FFY_INIT_ATTR.INIT0);
      jbits.setTileBits(4, 2, FFY_INIT_ATTR.CONFIG[2],
FFY_INIT_ATTR.INIT0);
      jbits.setTileBits(4, 2, FFY_SR_ATTR.CONFIG[0],
FFY_SR_ATTR.SRLOW);
      jbits.setTileBits(4, 2, FFY_SR_ATTR.CONFIG[1],
FFY_SR_ATTR.SRLOW);
      jbits.setTileBits(4, 2, FFY_SR_ATTR.CONFIG[2],
FFY_SR_ATTR.SRLOW);
      jbits.setTileBits(4, 2, FLUT.CONTENTS[0], new int[] {1, 1, 1, 1,
1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0});
      jbits.setTileBits(4, 2, FLUT.CONTENTS[1], new int[] {1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1});
      jbits.setTileBits(4, 2, FLUT.CONTENTS[2], new int[] {1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1});
      jbits.setTileBits(4, 2, FXMUX.CONFIG[0], FXMUX.FXOR);
      jbits.setTileBits(4, 2, GLUT.CONTENTS[0], new int[] {1, 1, 0, 0,
1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0});
      jbits.setTileBits(4, 2, GLUT.CONTENTS[1], new int[] {1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1});
      jbits.setTileBits(4, 2, GLUT.CONTENTS[2], new int[] {1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1});
      jbits.setTileBits(4, 2, GYMUX.CONFIG[0], GYMUX.GXOR);
      jbits.setTileBits(4, 2, SRINV.CONFIG[0], SRINV.SR);
      jbits.setTileBits(4, 2, SYNC_ATTR.CONFIG[0], SYNC_ATTR.SYNC);
      jbits.setTileBits(4, 2, BX1.BX1, BX1.N2BEG9);
      jbits.setTileBits(4, 2, BX3.BX3, BX3.BY3);
      jbits.setTileBits(4, 2, BY1.BY1, BY1.BX1);
      jbits.setTileBits(4, 2, BY2.BY2, BY2.BX3);
      jbits.setTileBits(4, 2, BY3.BY3, BY3.BX1);
      jbits.setTileBits(4, 2, CE_B1.CE_B1, CE_B1.W2MID8);
      jbits.setTileBits(4, 2, CE_B2.CE_B2, CE_B2.E2BEG9);
      jbits.setTileBits(4, 2, CLB_BY_S1.CLB_BY_S1,
CLB_BY_S1.CLB_BY_B1);
      jbits.setTileBits(4, 2, CLB_CE_S0.CLB_CE_S0, CLB_CE_S0.CLB_CE0);
      jbits.setTileBits(4, 2, CLB_CE_S1.CLB_CE_S1, CLB_CE_S1.CLB_CE1);
      jbits.setTileBits(4, 2, CLB_CE_S2.CLB_CE_S2, CLB_CE_S2.CLB_CE2);
      jbits.setTileBits(4, 2, CLB_CY0F_S0.CLB_CY0F_S0,
CLB_CY0F_S0.CLB_ZERO);
      jbits.setTileBits(4, 2, CLB_CYINIT_S0.CLB_CYINIT_S0,
CLB_CYINIT_S0.CLB_CIN_S0);
      jbits.setTileBits(4, 2, CLB_CYSELF_S0.CLB_CYSELF_S0,
CLB_CYSELF_S0.CLB_F_D_S0);
      jbits.setTileBits(4, 2, CLB_SR_S0.CLB_SR_S0, CLB_SR_S0.SR0);
      jbits.setTileBits(4, 2, CLB_XQ_D_S0.CLB_XQ_D_S0, CLB_XQ_D_S0.X0);
```

```
      jbits.setTileBits(4, 2, CLB_XQ_RESET_S0.CLB_XQ_RESET_S0,
CLB_XQ_RESET_S0.CLB_SRFF_S0);
      jbits.setTileBits(4, 2, CLB_XQ_SET_S0.CLB_XQ_SET_S0,
CLB_XQ_SET_S0.CLB_REVFF_S0);
      jbits.setTileBits(4, 2, CLB_YQ_D_S0.CLB_YQ_D_S0, CLB_YQ_D_S0.Y0);
      jbits.setTileBits(4, 2, CLB_YQ_RESET_S0.CLB_YQ_RESET_S0,
CLB_YQ_RESET_S0.CLB_SRFF_S0);
      jbits.setTileBits(4, 2, CLB_YQ_RESET_S1.CLB_YQ_RESET_S1,
CLB_YQ_RESET_S1.CLB_SRFF_S1);
      jbits.setTileBits(4, 2, CLB_YQ_RESET_S2.CLB_YQ_RESET_S2,
CLB_YQ_RESET_S2.CLB_SRFF_S2);
      jbits.setTileBits(4, 2, CLB_YQ_SET_S0.CLB_YQ_SET_S0,
CLB_YQ_SET_S0.CLB_REVFF_S0);
      jbits.setTileBits(4, 2, CLB_YQ_SET_S1.CLB_YQ_SET_S1,
CLB_YQ_SET_S1.CLB_REVFF_S1);
      jbits.setTileBits(4, 2, CLB_YQ_SET_S2.CLB_YQ_SET_S2,
CLB_YQ_SET_S2.CLB_REVFF_S2);
      jbits.setTileBits(4, 2, CLK0.CLK0, CLK0.GCLK4);
      jbits.setTileBits(4, 2, CLK1.CLK1, CLK1.GCLK4);
      jbits.setTileBits(4, 2, CLK2.CLK2, CLK2.GCLK4);
      jbits.setTileBits(4, 2, E2BEG0.E2BEG0, E2BEG0.N6END0);
      jbits.setTileBits(4, 2, E2BEG9.E2BEG9, E2BEG9.OMUX_N15);
      jbits.setTileBits(4, 2, F4_B0.F4_B0, F4_B0.OMUX2);
      jbits.setTileBits(4, 2, G2_B0.G2_B0, G2_B0.OMUX6);
      jbits.setTileBits(4, 2, LH12.LH12, LH12.OMUX11);
      jbits.setTileBits(4, 2, N2BEG9.N2BEG9, N2BEG9.OMUX15);
      jbits.setTileBits(4, 2, OMUX0.OMUX0, OMUX0.YQ0);
      jbits.setTileBits(4, 2, OMUX11.OMUX11, OMUX11.YQ2);
      jbits.setTileBits(4, 2, OMUX15.OMUX15, OMUX15.YQ1);
      jbits.setTileBits(4, 2, OMUX2.OMUX2, OMUX2.XQ0);
      jbits.setTileBits(4, 2, OMUX4.OMUX4, OMUX4.XQ0);
      jbits.setTileBits(4, 2, OMUX6.OMUX6, OMUX6.YQ0);
      jbits.setTileBits(4, 2, SR0.SR0, SR0.E2BEG0);
      jbits.setTileBits(4, 2, X0.X0, X0.CLB_XORF_S0);
      jbits.setTileBits(4, 2, Y0.Y0, Y0.CLB_XORG_S0);

      // Configuration for the Center tile at (3, 2)
      jbits.setTileBits(3, 2, CEINV.CONFIG[0], CEINV.CE);
      jbits.setTileBits(3, 2, CEINV.CONFIG[1], CEINV.CE);
      jbits.setTileBits(3, 2, CY0F.CONFIG[0], CY0F.ZERO);
      jbits.setTileBits(3, 2, CY0F.CONFIG[1], CY0F.ZERO);
      jbits.setTileBits(3, 2, CY0G.CONFIG[0], CY0G.ZERO);
      jbits.setTileBits(3, 2, CY0G.CONFIG[1], CY0G.ZERO);
      jbits.setTileBits(3, 2, CYINIT.CONFIG[0], CYINIT.CIN);
      jbits.setTileBits(3, 2, CYINIT.CONFIG[1], CYINIT.CIN);
      jbits.setTileBits(3, 2, CYSELF.CONFIG[0], CYSELF.F);
      jbits.setTileBits(3, 2, CYSELF.CONFIG[1], CYSELF.F);
      jbits.setTileBits(3, 2, CYSELG.CONFIG[0], CYSELG.G);
      jbits.setTileBits(3, 2, CYSELG.CONFIG[1], CYSELG.G);
      jbits.setTileBits(3, 2, DXMUX.CONFIG[0], DXMUX.DX);
      jbits.setTileBits(3, 2, DXMUX.CONFIG[1], DXMUX.DX);
      jbits.setTileBits(3, 2, DYMUX.CONFIG[0], DYMUX.DY);
      jbits.setTileBits(3, 2, DYMUX.CONFIG[1], DYMUX.DY);
      jbits.setTileBits(3, 2, FFX_INIT_ATTR.CONFIG[0],
FFX_INIT_ATTR.INIT0);
      jbits.setTileBits(3, 2, FFX_INIT_ATTR.CONFIG[1],
FFX_INIT_ATTR.INIT0);
```

```
      jbits.setTileBits(3, 2, FFX_SR_ATTR.CONFIG[0],
FFX_SR_ATTR.SRLOW);
      jbits.setTileBits(3, 2, FFX_SR_ATTR.CONFIG[1],
FFX_SR_ATTR.SRLOW);
      jbits.setTileBits(3, 2, FFY_INIT_ATTR.CONFIG[0],
FFY_INIT_ATTR.INIT0);
      jbits.setTileBits(3, 2, FFY_INIT_ATTR.CONFIG[1],
FFY_INIT_ATTR.INIT0);
      jbits.setTileBits(3, 2, FFY_SR_ATTR.CONFIG[0],
FFY_SR_ATTR.SRLOW);
      jbits.setTileBits(3, 2, FFY_SR_ATTR.CONFIG[1],
FFY_SR_ATTR.SRLOW);
      jbits.setTileBits(3, 2, FLUT.CONTENTS[0], new int[] {1, 1, 0, 0,
1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0});
      jbits.setTileBits(3, 2, FLUT.CONTENTS[1], new int[] {1, 0, 1, 0,
1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0});
      jbits.setTileBits(3, 2, FLUT.CONTENTS[2], new int[] {1, 1, 1, 1,
1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1});
      jbits.setTileBits(3, 2, FXMUX.CONFIG[0], FXMUX.FXOR);
      jbits.setTileBits(3, 2, FXMUX.CONFIG[1], FXMUX.FXOR);
      jbits.setTileBits(3, 2, GLUT.CONTENTS[0], new int[] {1, 1, 1, 1,
1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0});
      jbits.setTileBits(3, 2, GLUT.CONTENTS[1], new int[] {1, 1, 0, 0,
1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0});
      jbits.setTileBits(3, 2, GLUT.CONTENTS[2], new int[] {1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0});
      jbits.setTileBits(3, 2, GYMUX.CONFIG[0], GYMUX.GXOR);
      jbits.setTileBits(3, 2, GYMUX.CONFIG[1], GYMUX.GXOR);
      jbits.setTileBits(3, 2, SRINV.CONFIG[0], SRINV.SR);
      jbits.setTileBits(3, 2, SRINV.CONFIG[1], SRINV.SR);
      jbits.setTileBits(3, 2, SYNC_ATTR.CONFIG[0], SYNC_ATTR.SYNC);
      jbits.setTileBits(3, 2, SYNC_ATTR.CONFIG[1], SYNC_ATTR.SYNC);
      jbits.setTileBits(3, 2, BX0.BX0, BX0.S2BEG2);
      jbits.setTileBits(3, 2, BY1.BY1, BY1.E2BEG7);
      jbits.setTileBits(3, 2, BY3.BY3, BY3.BX0);
      jbits.setTileBits(3, 2, CLB_CE_S0.CLB_CE_S0, CLB_CE_S0.CLB_CE0);
      jbits.setTileBits(3, 2, CLB_CE_S1.CLB_CE_S1, CLB_CE_S1.CLB_CE1);
      jbits.setTileBits(3, 2, CLB_CY0F_S0.CLB_CY0F_S0,
CLB_CY0F_S0.CLB_ZERO);
      jbits.setTileBits(3, 2, CLB_CY0F_S1.CLB_CY0F_S1,
CLB_CY0F_S1.CLB_ZERO);
      jbits.setTileBits(3, 2, CLB_CY0G_S0.CLB_CY0G_S0,
CLB_CY0G_S0.CLB_ZERO);
      jbits.setTileBits(3, 2, CLB_CY0G_S1.CLB_CY0G_S1,
CLB_CY0G_S1.CLB_ZERO);
      jbits.setTileBits(3, 2, CLB_CYINIT_S0.CLB_CYINIT_S0,
CLB_CYINIT_S0.CLB_CIN_S0);
      jbits.setTileBits(3, 2, CLB_CYINIT_S1.CLB_CYINIT_S1,
CLB_CYINIT_S1.CLB_CIN_S1);
      jbits.setTileBits(3, 2, CLB_CYSELF_S0.CLB_CYSELF_S0,
CLB_CYSELF_S0.CLB_F_D_S0);
      jbits.setTileBits(3, 2, CLB_CYSELF_S1.CLB_CYSELF_S1,
CLB_CYSELF_S1.CLB_F_D_S1);
      jbits.setTileBits(3, 2, CLB_CYSELG_S0.CLB_CYSELG_S0,
CLB_CYSELG_S0.CLB_G_D_S0);
      jbits.setTileBits(3, 2, CLB_CYSELG_S1.CLB_CYSELG_S1,
CLB_CYSELG_S1.CLB_G_D_S1);
```

```
        jbits.setTileBits(3, 2, CLB_SR_S0.CLB_SR_S0, CLB_SR_S0.SR0);
        jbits.setTileBits(3, 2, CLB_SR_S1.CLB_SR_S1, CLB_SR_S1.SR1);
        jbits.setTileBits(3, 2, CLB_XQ_D_S0.CLB_XQ_D_S0, CLB_XQ_D_S0.X0);
        jbits.setTileBits(3, 2, CLB_XQ_D_S1.CLB_XQ_D_S1, CLB_XQ_D_S1.X1);
        jbits.setTileBits(3, 2, CLB_XQ_RESET_S0.CLB_XQ_RESET_S0,
CLB_XQ_RESET_S0.CLB_SRFF_S0);
        jbits.setTileBits(3, 2, CLB_XQ_RESET_S1.CLB_XQ_RESET_S1,
CLB_XQ_RESET_S1.CLB_SRFF_S1);
        jbits.setTileBits(3, 2, CLB_XQ_SET_S0.CLB_XQ_SET_S0,
CLB_XQ_SET_S0.CLB_REVFF_S0);
        jbits.setTileBits(3, 2, CLB_XQ_SET_S1.CLB_XQ_SET_S1,
CLB_XQ_SET_S1.CLB_REVFF_S1);
        jbits.setTileBits(3, 2, CLB_YQ_D_S0.CLB_YQ_D_S0, CLB_YQ_D_S0.Y0);
        jbits.setTileBits(3, 2, CLB_YQ_D_S1.CLB_YQ_D_S1, CLB_YQ_D_S1.Y1);
        jbits.setTileBits(3, 2, CLB_YQ_RESET_S0.CLB_YQ_RESET_S0,
CLB_YQ_RESET_S0.CLB_SRFF_S0);
        jbits.setTileBits(3, 2, CLB_YQ_RESET_S1.CLB_YQ_RESET_S1,
CLB_YQ_RESET_S1.CLB_SRFF_S1);
        jbits.setTileBits(3, 2, CLB_YQ_SET_S0.CLB_YQ_SET_S0,
CLB_YQ_SET_S0.CLB_REVFF_S0);
        jbits.setTileBits(3, 2, CLB_YQ_SET_S1.CLB_YQ_SET_S1,
CLB_YQ_SET_S1.CLB_REVFF_S1);
        jbits.setTileBits(3, 2, CLK0.CLK0, CLK0.GCLK4);
        jbits.setTileBits(3, 2, CLK1.CLK1, CLK1.GCLK4);
        jbits.setTileBits(3, 2, E2BEG1.E2BEG1, E2BEG1.X2);
        jbits.setTileBits(3, 2, E2BEG7.E2BEG7, E2BEG7.Y2);
        jbits.setTileBits(3, 2, F1_B1.F1_B1, F1_B1.OMUX2);
        jbits.setTileBits(3, 2, F1_B2.F1_B2, F1_B2.OMUX_N15);
        jbits.setTileBits(3, 2, F2_B0.F2_B0, F2_B0.OMUX6);
        jbits.setTileBits(3, 2, F2_B2.F2_B2, F2_B2.OMUX_S4);
        jbits.setTileBits(3, 2, F3_B2.F3_B2, F3_B2.OMUX9);
        jbits.setTileBits(3, 2, F4_B2.F4_B2, F4_B2.BY1);
        jbits.setTileBits(3, 2, G1_B2.G1_B2, G1_B2.BY3);
        jbits.setTileBits(3, 2, G2_B1.G2_B1, G2_B1.OMUX9);
        jbits.setTileBits(3, 2, G2_B2.G2_B2, G2_B2.OMUX6);
        jbits.setTileBits(3, 2, G3_B2.G3_B2, G3_B2.OMUX_N11);
        jbits.setTileBits(3, 2, G4_B0.G4_B0, G4_B0.S2BEG2);
        jbits.setTileBits(3, 2, G4_B2.G4_B2, G4_B2.OMUX2);
        jbits.setTileBits(3, 2, OMUX0.OMUX0, OMUX0.X2);
        jbits.setTileBits(3, 2, OMUX13.OMUX13, OMUX13.X2);
        jbits.setTileBits(3, 2, OMUX15.OMUX15, OMUX15.X2);
        jbits.setTileBits(3, 2, OMUX2.OMUX2, OMUX2.XQ1);
        jbits.setTileBits(3, 2, OMUX4.OMUX4, OMUX4.YQ0);
        jbits.setTileBits(3, 2, OMUX6.OMUX6, OMUX6.XQ0);
        jbits.setTileBits(3, 2, OMUX9.OMUX9, OMUX9.YQ1);
        jbits.setTileBits(3, 2, S2BEG0.S2BEG0, S2BEG0.OMUX_S0);
        jbits.setTileBits(3, 2, S2BEG2.S2BEG2, S2BEG2.OMUX4);
        jbits.setTileBits(3, 2, S6BEG0.S6BEG0, S6BEG0.OMUX0);
        jbits.setTileBits(3, 2, SR0.SR0, SR0.N6D0);
        jbits.setTileBits(3, 2, SR1.SR1, SR1.E2BEG1);
        jbits.setTileBits(3, 2, X0.X0, X0.CLB_XORF_S0);
        jbits.setTileBits(3, 2, X1.X1, X1.CLB_XORF_S1);
        jbits.setTileBits(3, 2, Y0.Y0, Y0.CLB_XORG_S0);
        jbits.setTileBits(3, 2, Y1.Y1, Y1.CLB_XORG_S1);

        // Configuration for the Center tile at (2, 2)
        jbits.setTileBits(2, 2, BXINV.CONFIG[0], BXINV.BX_B);
```

```
        jbits.setTileBits(2, 2, CEINV.CONFIG[0], CEINV.CE);
        jbits.setTileBits(2, 2, CEINV.CONFIG[1], CEINV.CE);
        jbits.setTileBits(2, 2, CY0F.CONFIG[0], CY0F.ONE);
        jbits.setTileBits(2, 2, CY0F.CONFIG[1], CY0F.ZERO);
        jbits.setTileBits(2, 2, CY0G.CONFIG[0], CY0G.ZERO);
        jbits.setTileBits(2, 2, CY0G.CONFIG[1], CY0G.ZERO);
        jbits.setTileBits(2, 2, CYINIT.CONFIG[1], CYINIT.CIN);
        jbits.setTileBits(2, 2, CYSELF.CONFIG[0], CYSELF.F);
        jbits.setTileBits(2, 2, CYSELF.CONFIG[1], CYSELF.F);
        jbits.setTileBits(2, 2, CYSELG.CONFIG[0], CYSELG.G);
        jbits.setTileBits(2, 2, CYSELG.CONFIG[1], CYSELG.G);
        jbits.setTileBits(2, 2, DXMUX.CONFIG[0], DXMUX.DX);
        jbits.setTileBits(2, 2, DXMUX.CONFIG[1], DXMUX.DX);
        jbits.setTileBits(2, 2, DYMUX.CONFIG[0], DYMUX.DY);
        jbits.setTileBits(2, 2, DYMUX.CONFIG[1], DYMUX.DY);
        jbits.setTileBits(2, 2, FFX_INIT_ATTR.CONFIG[0],
FFX_INIT_ATTR.INIT0);
        jbits.setTileBits(2, 2, FFX_INIT_ATTR.CONFIG[1],
FFX_INIT_ATTR.INIT0);
        jbits.setTileBits(2, 2, FFX_SR_ATTR.CONFIG[0],
FFX_SR_ATTR.SRLOW);
        jbits.setTileBits(2, 2, FFX_SR_ATTR.CONFIG[1],
FFX_SR_ATTR.SRLOW);
        jbits.setTileBits(2, 2, FFY_INIT_ATTR.CONFIG[0],
FFY_INIT_ATTR.INIT0);
        jbits.setTileBits(2, 2, FFY_INIT_ATTR.CONFIG[1],
FFY_INIT_ATTR.INIT0);
        jbits.setTileBits(2, 2, FFY_SR_ATTR.CONFIG[0],
FFY_SR_ATTR.SRLOW);
        jbits.setTileBits(2, 2, FFY_SR_ATTR.CONFIG[1],
FFY_SR_ATTR.SRLOW);
        jbits.setTileBits(2, 2, FLUT.CONTENTS[0], new int[] {0, 0, 0, 0,
0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1});
        jbits.setTileBits(2, 2, FLUT.CONTENTS[1], new int[] {1, 1, 1, 1,
1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0});
        jbits.setTileBits(2, 2, FLUT.CONTENTS[2], new int[] {1, 1, 1, 1,
1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1});
        jbits.setTileBits(2, 2, FXMUX.CONFIG[1], FXMUX.FXOR);
        jbits.setTileBits(2, 2, GLUT.CONTENTS[0], new int[] {1, 1, 0, 0,
1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0});
        jbits.setTileBits(2, 2, GLUT.CONTENTS[1], new int[] {1, 1, 0, 0,
1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0});
        jbits.setTileBits(2, 2, GLUT.CONTENTS[2], new int[] {1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1});
        jbits.setTileBits(2, 2, GYMUX.CONFIG[0], GYMUX.GXOR);
        jbits.setTileBits(2, 2, GYMUX.CONFIG[1], GYMUX.GXOR);
        jbits.setTileBits(2, 2, SRINV.CONFIG[0], SRINV.SR);
        jbits.setTileBits(2, 2, SRINV.CONFIG[1], SRINV.SR);
        jbits.setTileBits(2, 2, SYNC_ATTR.CONFIG[0], SYNC_ATTR.SYNC);
        jbits.setTileBits(2, 2, SYNC_ATTR.CONFIG[1], SYNC_ATTR.SYNC);
        jbits.setTileBits(2, 2, CLB_BX_S0.CLB_BX_S0,
CLB_BX_S0.CLB_BX_B0);
        jbits.setTileBits(2, 2, CLB_CE_S0.CLB_CE_S0, CLB_CE_S0.CLB_CE0);
        jbits.setTileBits(2, 2, CLB_CE_S1.CLB_CE_S1, CLB_CE_S1.CLB_CE1);
        jbits.setTileBits(2, 2, CLB_CY0F_S0.CLB_CY0F_S0,
CLB_CY0F_S0.CLB_ONE);
```

```
      jbits.setTileBits(2, 2, CLB_CY0F_S1.CLB_CY0F_S1,
CLB_CY0F_S1.CLB_ZERO);
      jbits.setTileBits(2, 2, CLB_CY0G_S0.CLB_CY0G_S0,
CLB_CY0G_S0.CLB_ZERO);
      jbits.setTileBits(2, 2, CLB_CY0G_S1.CLB_CY0G_S1,
CLB_CY0G_S1.CLB_ZERO);
      jbits.setTileBits(2, 2, CLB_CYINIT_S1.CLB_CYINIT_S1,
CLB_CYINIT_S1.CLB_CIN_S1);
      jbits.setTileBits(2, 2, CLB_CYSELF_S0.CLB_CYSELF_S0,
CLB_CYSELF_S0.CLB_F_D_S0);
      jbits.setTileBits(2, 2, CLB_CYSELF_S1.CLB_CYSELF_S1,
CLB_CYSELF_S1.CLB_F_D_S1);
      jbits.setTileBits(2, 2, CLB_CYSELG_S0.CLB_CYSELG_S0,
CLB_CYSELG_S0.CLB_G_D_S0);
      jbits.setTileBits(2, 2, CLB_CYSELG_S1.CLB_CYSELG_S1,
CLB_CYSELG_S1.CLB_G_D_S1);
      jbits.setTileBits(2, 2, CLB_SR_S0.CLB_SR_S0, CLB_SR_S0.SR0);
      jbits.setTileBits(2, 2, CLB_SR_S1.CLB_SR_S1, CLB_SR_S1.SR1);
      jbits.setTileBits(2, 2, CLB_XQ_D_S0.CLB_XQ_D_S0, CLB_XQ_D_S0.X0);
      jbits.setTileBits(2, 2, CLB_XQ_D_S1.CLB_XQ_D_S1, CLB_XQ_D_S1.X1);
      jbits.setTileBits(2, 2, CLB_XQ_RESET_S0.CLB_XQ_RESET_S0,
CLB_XQ_RESET_S0.CLB_SRFF_S0);
      jbits.setTileBits(2, 2, CLB_XQ_RESET_S1.CLB_XQ_RESET_S1,
CLB_XQ_RESET_S1.CLB_SRFF_S1);
      jbits.setTileBits(2, 2, CLB_XQ_SET_S0.CLB_XQ_SET_S0,
CLB_XQ_SET_S0.CLB_REVFF_S0);
      jbits.setTileBits(2, 2, CLB_XQ_SET_S1.CLB_XQ_SET_S1,
CLB_XQ_SET_S1.CLB_REVFF_S1);
      jbits.setTileBits(2, 2, CLB_YQ_D_S0.CLB_YQ_D_S0, CLB_YQ_D_S0.Y0);
      jbits.setTileBits(2, 2, CLB_YQ_D_S1.CLB_YQ_D_S1, CLB_YQ_D_S1.Y1);
      jbits.setTileBits(2, 2, CLB_YQ_RESET_S0.CLB_YQ_RESET_S0,
CLB_YQ_RESET_S0.CLB_SRFF_S0);
      jbits.setTileBits(2, 2, CLB_YQ_RESET_S1.CLB_YQ_RESET_S1,
CLB_YQ_RESET_S1.CLB_SRFF_S1);
      jbits.setTileBits(2, 2, CLB_YQ_SET_S0.CLB_YQ_SET_S0,
CLB_YQ_SET_S0.CLB_REVFF_S0);
      jbits.setTileBits(2, 2, CLB_YQ_SET_S1.CLB_YQ_SET_S1,
CLB_YQ_SET_S1.CLB_REVFF_S1);
      jbits.setTileBits(2, 2, CLK0.CLK0, CLK0.GCLK4);
      jbits.setTileBits(2, 2, CLK1.CLK1, CLK1.GCLK4);
      jbits.setTileBits(2, 2, F1_B2.F1_B2, F1_B2.OMUX13);
      jbits.setTileBits(2, 2, F2_B2.F2_B2, F2_B2.OMUX6);
      jbits.setTileBits(2, 2, F3_B2.F3_B2, F3_B2.OMUX9);
      jbits.setTileBits(2, 2, F4_B0.F4_B0, F4_B0.OMUX2);
      jbits.setTileBits(2, 2, F4_B1.F4_B1, F4_B1.OMUX13);
      jbits.setTileBits(2, 2, F4_B2.F4_B2, F4_B2.S2MID0);
      jbits.setTileBits(2, 2, G2_B0.G2_B0, G2_B0.OMUX6);
      jbits.setTileBits(2, 2, G2_B1.G2_B1, G2_B1.OMUX9);
      jbits.setTileBits(2, 2, OMUX11.OMUX11, OMUX11.XQ0);
      jbits.setTileBits(2, 2, OMUX13.OMUX13, OMUX13.XQ1);
      jbits.setTileBits(2, 2, OMUX15.OMUX15, OMUX15.X2);
      jbits.setTileBits(2, 2, OMUX2.OMUX2, OMUX2.XQ0);
      jbits.setTileBits(2, 2, OMUX6.OMUX6, OMUX6.YQ0);
      jbits.setTileBits(2, 2, OMUX9.OMUX9, OMUX9.YQ1);
      jbits.setTileBits(2, 2, SR0.SR0, SR0.S6A0);
      jbits.setTileBits(2, 2, SR1.SR1, SR1.S6A0);
      jbits.setTileBits(2, 2, X1.X1, X1.CLB_XORF_S1);
```

```
        jbits.setTileBits(2, 2, Y0.Y0, Y0.CLB_XORG_S0);
        jbits.setTileBits(2, 2, Y1.Y1, Y1.CLB_XORG_S1);

        // Configuration for the Center tile at (4, 3)
        jbits.setTileBits(4, 3, W2BEG8.W2BEG8, W2BEG8.N2MID8);

        // Configuration for the Center tile at (3, 3)
        jbits.setTileBits(3, 3, N2BEG8.N2BEG8, N2BEG8.OMUX_E13);

        // Configuration for the Center tile at (33, 9)
        jbits.setTileBits(33, 9, W6BEG4.W6BEG4, W6BEG4.W6END2);

        // Configuration for the Center tile at (47, 11)
        jbits.setTileBits(47, 11, N2BEG6.N2BEG6, N2BEG6.W6END5);

        // Configuration for the Center tile at (33, 15)
        jbits.setTileBits(33, 15, W6BEG2.W6BEG2, W6BEG2.W6END0);

        // Configuration for the Center tile at (48, 16)
        jbits.setTileBits(48, 16, E6BEG8.E6BEG8, E6BEG8.E6END_S0);

        // Configuration for the Center tile at (47, 17)
        jbits.setTileBits(47, 17, W6BEG5.W6BEG5, W6BEG5.W6END5);

        // Configuration for the Center tile at (42, 23)
        jbits.setTileBits(42, 23, FLUT.CONTENTS[3], new int[] {1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1});
        jbits.setTileBits(42, 23, GLUT.CONTENTS[3], new int[] {1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1});
        jbits.setTileBits(42, 23, OMUX7.OMUX7, OMUX7.Y3);

        // Configuration for the Center tile at (33, 23)
        jbits.setTileBits(33, 23, W6BEG0.W6BEG0, W6BEG0.W6END_N8);

        // Configuration for the Center tile at (5, 23)
        jbits.setTileBits(5, 23, E6BEG1.E6BEG1, E6BEG1.LH0);

        // Configuration for the Center tile at (48, 24)
        jbits.setTileBits(48, 24, S6BEG6.S6BEG6, S6BEG6.LV0);
        jbits.setTileBits(48, 24, S6BEG7.S6BEG7, S6BEG7.E6END8);

        // Configuration for the Center tile at (42, 24)
        jbits.setTileBits(42, 24, CEINV.CONFIG[0], CEINV.CE);
        jbits.setTileBits(42, 24, CEINV.CONFIG[2], CEINV.CE);
        jbits.setTileBits(42, 24, DIG_MUX.CONFIG[0], DIG_MUX.BY);
        jbits.setTileBits(42, 24, DYMUX.CONFIG[0], DYMUX.DY);
        jbits.setTileBits(42, 24, FFY_INIT_ATTR.CONFIG[0],
FFY_INIT_ATTR.INIT0);
        jbits.setTileBits(42, 24, FFY_INIT_ATTR.CONFIG[2],
FFY_INIT_ATTR.INIT0);
        jbits.setTileBits(42, 24, FFY_SR_ATTR.CONFIG[0],
FFY_SR_ATTR.SRLOW);
        jbits.setTileBits(42, 24, FFY_SR_ATTR.CONFIG[2],
FFY_SR_ATTR.SRLOW);
        jbits.setTileBits(42, 24, FLUT.CONTENTS[0], new int[] {1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1});
```

```
        jbits.setTileBits(42, 24, FLUT.CONTENTS[2], new int[] {1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1});
        jbits.setTileBits(42, 24, FLUT.CONTENTS[3], new int[] {1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1});
        jbits.setTileBits(42, 24, GLUT.CONFIG[0], GLUT.RAM_SHIFT_REG);
        jbits.setTileBits(42, 24, GLUT.CONTENTS[0], new int[] {1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1});
        jbits.setTileBits(42, 24, GLUT.CONTENTS[2], new int[] {1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1});
        jbits.setTileBits(42, 24, GLUT.CONTENTS[3], new int[] {1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1});
        jbits.setTileBits(42, 24, SRINV.CONFIG[0], SRINV.SR);
        jbits.setTileBits(42, 24, BX1.BX1, BX1.E2BEG9);
        jbits.setTileBits(42, 24, BX2.BX2, BX2.BY1);
        jbits.setTileBits(42, 24, BX3.BX3, BX3.N2MID7);
        jbits.setTileBits(42, 24, BY0.BY0, BY0.BX3);
        jbits.setTileBits(42, 24, BY1.BY1, BY1.BX1);
        jbits.setTileBits(42, 24, BY2.BY2, BY2.BX2);
        jbits.setTileBits(42, 24, CE_B0.CE_B0, CE_B0.W2END8);
        jbits.setTileBits(42, 24, CE_B2.CE_B2, CE_B2.W2END9);
        jbits.setTileBits(42, 24, CLB_CE_S0.CLB_CE_S0,
CLB_CE_S0.CLB_CE0);
        jbits.setTileBits(42, 24, CLB_CE_S2.CLB_CE_S2,
CLB_CE_S2.CLB_CE2);
        jbits.setTileBits(42, 24, CLB_SRFF_S0.CLB_SRFF_S0,
CLB_SRFF_S0.OFF);
        jbits.setTileBits(42, 24, CLB_SR_S0.CLB_SR_S0, CLB_SR_S0.SR0);
        jbits.setTileBits(42, 24, CLB_WF1_B0.CLB_WF1_B0,
CLB_WF1_B0.F1_B0);
        jbits.setTileBits(42, 24, CLB_WF2_B0.CLB_WF2_B0,
CLB_WF2_B0.F2_B0);
        jbits.setTileBits(42, 24, CLB_WF3_B0.CLB_WF3_B0,
CLB_WF3_B0.F3_B0);
        jbits.setTileBits(42, 24, CLB_WF4_B0.CLB_WF4_B0,
CLB_WF4_B0.F4_B0);
        jbits.setTileBits(42, 24, CLB_WG1_B0.CLB_WG1_B0,
CLB_WG1_B0.G1_B0);
        jbits.setTileBits(42, 24, CLB_WG2_B0.CLB_WG2_B0,
CLB_WG2_B0.G2_B0);
        jbits.setTileBits(42, 24, CLB_WG3_B0.CLB_WG3_B0,
CLB_WG3_B0.G3_B0);
        jbits.setTileBits(42, 24, CLB_WG4_B0.CLB_WG4_B0,
CLB_WG4_B0.G4_B0);
        jbits.setTileBits(42, 24, CLB_YQ_D_S0.CLB_YQ_D_S0,
CLB_YQ_D_S0.Y0);
        jbits.setTileBits(42, 24, CLB_YQ_RESET_S0.CLB_YQ_RESET_S0,
CLB_YQ_RESET_S0.CLB_SRFF_S0);
        jbits.setTileBits(42, 24, CLB_YQ_RESET_S2.CLB_YQ_RESET_S2,
CLB_YQ_RESET_S2.CLB_SRFF_S2);
        jbits.setTileBits(42, 24, CLB_YQ_SET_S0.CLB_YQ_SET_S0,
CLB_YQ_SET_S0.CLB_REVFF_S0);
        jbits.setTileBits(42, 24, CLB_YQ_SET_S2.CLB_YQ_SET_S2,
CLB_YQ_SET_S2.CLB_REVFF_S2);
        jbits.setTileBits(42, 24, CLK0.CLK0, CLK0.N6B6);
        jbits.setTileBits(42, 24, CLK2.CLK2, CLK2.S6D6);
        jbits.setTileBits(42, 24, E2BEG9.E2BEG9, E2BEG9.OMUX15);
        jbits.setTileBits(42, 24, G2_B0.G2_B0, G2_B0.OMUX_E7);
```

```
        jbits.setTileBits(42, 24, G3_B0.G3_B0, G3_B0.OMUX_N11);
        jbits.setTileBits(42, 24, G4_B0.G4_B0, G4_B0.OMUX2);
        jbits.setTileBits(42, 24, OMUX15.OMUX15, OMUX15.YQ0);
        jbits.setTileBits(42, 24, OMUX2.OMUX2, OMUX2.Y3);
        jbits.setTileBits(42, 24, OMUX7.OMUX7, OMUX7.YQ2);
        jbits.setTileBits(42, 24, SR0.SR0, SR0.W2END0);

        // Configuration for the Center tile at (41, 24)
        jbits.setTileBits(41, 24, CEINV.CONFIG[0], CEINV.CE);
        jbits.setTileBits(41, 24, DXMUX.CONFIG[0], DXMUX.DX);
        jbits.setTileBits(41, 24, DYMUX.CONFIG[0], DYMUX.DY);
        jbits.setTileBits(41, 24, FFX_INIT_ATTR.CONFIG[0],
FFX_INIT_ATTR.INIT0);
        jbits.setTileBits(41, 24, FFX_SR_ATTR.CONFIG[0],
FFX_SR_ATTR.SRLOW);
        jbits.setTileBits(41, 24, FFY_INIT_ATTR.CONFIG[0],
FFY_INIT_ATTR.INIT0);
        jbits.setTileBits(41, 24, FFY_SR_ATTR.CONFIG[0],
FFY_SR_ATTR.SRLOW);
        jbits.setTileBits(41, 24, FLUT.CONTENTS[0], new int[] {1, 0, 1,
1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0});
        jbits.setTileBits(41, 24, FLUT.CONTENTS[3], new int[] {1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1});
        jbits.setTileBits(41, 24, GLUT.CONTENTS[0], new int[] {1, 1, 1,
0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0});
        jbits.setTileBits(41, 24, GLUT.CONTENTS[3], new int[] {1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1});
        jbits.setTileBits(41, 24, BX3.BX3, BX3.N2BEG7);
        jbits.setTileBits(41, 24, BY0.BY0, BY0.BX3);
        jbits.setTileBits(41, 24, CE_B0.CE_B0, CE_B0.E2BEG8);
        jbits.setTileBits(41, 24, CLB_CE_S0.CLB_CE_S0,
CLB_CE_S0.CLB_CE0);
        jbits.setTileBits(41, 24, CLB_XQ_D_S0.CLB_XQ_D_S0,
CLB_XQ_D_S0.X0);
        jbits.setTileBits(41, 24, CLB_XQ_RESET_S0.CLB_XQ_RESET_S0,
CLB_XQ_RESET_S0.CLB_SRFF_S0);
        jbits.setTileBits(41, 24, CLB_XQ_SET_S0.CLB_XQ_SET_S0,
CLB_XQ_SET_S0.CLB_REVFF_S0);
        jbits.setTileBits(41, 24, CLB_YQ_D_S0.CLB_YQ_D_S0,
CLB_YQ_D_S0.Y0);
        jbits.setTileBits(41, 24, CLB_YQ_RESET_S0.CLB_YQ_RESET_S0,
CLB_YQ_RESET_S0.CLB_SRFF_S0);
        jbits.setTileBits(41, 24, CLB_YQ_SET_S0.CLB_YQ_SET_S0,
CLB_YQ_SET_S0.CLB_REVFF_S0);
        jbits.setTileBits(41, 24, CLK0.CLK0, CLK0.S6END6);
        jbits.setTileBits(41, 24, E2BEG6.E2BEG6, E2BEG6.S6END7);
        jbits.setTileBits(41, 24, E2BEG8.E2BEG8, E2BEG8.N2MID8);
        jbits.setTileBits(41, 24, F1_B0.F1_B0, F1_B0.OMUX13);
        jbits.setTileBits(41, 24, F2_B0.F2_B0, F2_B0.W2END3);
        jbits.setTileBits(41, 24, F3_B0.F3_B0, F3_B0.N2MID6);
        jbits.setTileBits(41, 24, F4_B0.F4_B0, F4_B0.BY0);
        jbits.setTileBits(41, 24, G1_B0.G1_B0, G1_B0.OMUX13);
        jbits.setTileBits(41, 24, G2_B0.G2_B0, G2_B0.W2END3);
        jbits.setTileBits(41, 24, G3_B0.G3_B0, G3_B0.N2MID6);
        jbits.setTileBits(41, 24, G4_B0.G4_B0, G4_B0.OMUX2);
        jbits.setTileBits(41, 24, N2BEG7.N2BEG7, N2BEG7.S6END7);
        jbits.setTileBits(41, 24, OMUX11.OMUX11, OMUX11.Y3);
```

```
        jbits.setTileBits(41, 24, OMUX13.OMUX13, OMUX13.XQ0);
        jbits.setTileBits(41, 24, OMUX2.OMUX2, OMUX2.YQ0);
        jbits.setTileBits(41, 24, OMUX4.OMUX4, OMUX4.YQ0);
        jbits.setTileBits(41, 24, S2BEG5.S2BEG5, S2BEG5.W2END3);

        // Configuration for the Center tile at (40, 24)
        jbits.setTileBits(40, 24, CEINV.CONFIG[0], CEINV.CE);
        jbits.setTileBits(40, 24, CEINV.CONFIG[1], CEINV.CE);
        jbits.setTileBits(40, 24, CEINV.CONFIG[2], CEINV.CE);
        jbits.setTileBits(40, 24, DXMUX.CONFIG[0], DXMUX.DX);
        jbits.setTileBits(40, 24, DXMUX.CONFIG[1], DXMUX.DX);
        jbits.setTileBits(40, 24, DXMUX.CONFIG[2], DXMUX.DX);
        jbits.setTileBits(40, 24, DYMUX.CONFIG[0], DYMUX.DY);
        jbits.setTileBits(40, 24, DYMUX.CONFIG[1], DYMUX.DY);
        jbits.setTileBits(40, 24, DYMUX.CONFIG[2], DYMUX.DY);
        jbits.setTileBits(40, 24, FFX_INIT_ATTR.CONFIG[0],
FFX_INIT_ATTR.INIT0);
        jbits.setTileBits(40, 24, FFX_INIT_ATTR.CONFIG[1],
FFX_INIT_ATTR.INIT0);
        jbits.setTileBits(40, 24, FFX_INIT_ATTR.CONFIG[2],
FFX_INIT_ATTR.INIT0);
        jbits.setTileBits(40, 24, FFX_SR_ATTR.CONFIG[0],
FFX_SR_ATTR.SRLOW);
        jbits.setTileBits(40, 24, FFX_SR_ATTR.CONFIG[1],
FFX_SR_ATTR.SRLOW);
        jbits.setTileBits(40, 24, FFX_SR_ATTR.CONFIG[2],
FFX_SR_ATTR.SRLOW);
        jbits.setTileBits(40, 24, FFY_INIT_ATTR.CONFIG[0],
FFY_INIT_ATTR.INIT0);
        jbits.setTileBits(40, 24, FFY_INIT_ATTR.CONFIG[1],
FFY_INIT_ATTR.INIT0);
        jbits.setTileBits(40, 24, FFY_INIT_ATTR.CONFIG[2],
FFY_INIT_ATTR.INIT0);
        jbits.setTileBits(40, 24, FFY_SR_ATTR.CONFIG[0],
FFY_SR_ATTR.SRLOW);
        jbits.setTileBits(40, 24, FFY_SR_ATTR.CONFIG[1],
FFY_SR_ATTR.SRLOW);
        jbits.setTileBits(40, 24, FFY_SR_ATTR.CONFIG[2],
FFY_SR_ATTR.SRLOW);
        jbits.setTileBits(40, 24, FLUT.CONTENTS[0], new int[] {1, 1, 0,
0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0});
        jbits.setTileBits(40, 24, FLUT.CONTENTS[1], new int[] {1, 1, 1,
0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0});
        jbits.setTileBits(40, 24, FLUT.CONTENTS[2], new int[] {1, 1, 1,
1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0});
        jbits.setTileBits(40, 24, FLUT.CONTENTS[3], new int[] {1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1});
        jbits.setTileBits(40, 24, GLUT.CONTENTS[0], new int[] {1, 1, 1,
1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0});
        jbits.setTileBits(40, 24, GLUT.CONTENTS[1], new int[] {1, 0, 1,
1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0});
        jbits.setTileBits(40, 24, GLUT.CONTENTS[2], new int[] {1, 0, 1,
0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0});
        jbits.setTileBits(40, 24, GLUT.CONTENTS[3], new int[] {1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0});
        jbits.setTileBits(40, 24, BX0.BX0, BX0.S2BEG0);
        jbits.setTileBits(40, 24, BX3.BX3, BX3.N2BEG7);
```

```
        jbits.setTileBits(40, 24, BY0.BY0, BY0.BX3);
        jbits.setTileBits(40, 24, BY1.BY1, BY1.S2MID5);
        jbits.setTileBits(40, 24, BY2.BY2, BY2.W2END4);
        jbits.setTileBits(40, 24, BY3.BY3, BY3.BX0);
        jbits.setTileBits(40, 24, CE_B0.CE_B0, CE_B0.E2BEG8);
        jbits.setTileBits(40, 24, CE_B1.CE_B1, CE_B1.E2BEG8);
        jbits.setTileBits(40, 24, CE_B2.CE_B2, CE_B2.E2BEG9);
        jbits.setTileBits(40, 24, CLB_CE_S0.CLB_CE_S0,
CLB_CE_S0.CLB_CE0);
        jbits.setTileBits(40, 24, CLB_CE_S1.CLB_CE_S1,
CLB_CE_S1.CLB_CE1);
        jbits.setTileBits(40, 24, CLB_CE_S2.CLB_CE_S2,
CLB_CE_S2.CLB_CE2);
        jbits.setTileBits(40, 24, CLB_XQ_D_S0.CLB_XQ_D_S0,
CLB_XQ_D_S0.X0);
        jbits.setTileBits(40, 24, CLB_XQ_D_S1.CLB_XQ_D_S1,
CLB_XQ_D_S1.X1);
        jbits.setTileBits(40, 24, CLB_XQ_D_S2.CLB_XQ_D_S2,
CLB_XQ_D_S2.X2);
        jbits.setTileBits(40, 24, CLB_XQ_RESET_S0.CLB_XQ_RESET_S0,
CLB_XQ_RESET_S0.CLB_SRFF_S0);
        jbits.setTileBits(40, 24, CLB_XQ_RESET_S1.CLB_XQ_RESET_S1,
CLB_XQ_RESET_S1.CLB_SRFF_S1);
        jbits.setTileBits(40, 24, CLB_XQ_RESET_S2.CLB_XQ_RESET_S2,
CLB_XQ_RESET_S2.CLB_SRFF_S2);
        jbits.setTileBits(40, 24, CLB_XQ_SET_S0.CLB_XQ_SET_S0,
CLB_XQ_SET_S0.CLB_REVFF_S0);
        jbits.setTileBits(40, 24, CLB_XQ_SET_S1.CLB_XQ_SET_S1,
CLB_XQ_SET_S1.CLB_REVFF_S1);
        jbits.setTileBits(40, 24, CLB_XQ_SET_S2.CLB_XQ_SET_S2,
CLB_XQ_SET_S2.CLB_REVFF_S2);
        jbits.setTileBits(40, 24, CLB_YQ_D_S0.CLB_YQ_D_S0,
CLB_YQ_D_S0.Y0);
        jbits.setTileBits(40, 24, CLB_YQ_D_S1.CLB_YQ_D_S1,
CLB_YQ_D_S1.Y1);
        jbits.setTileBits(40, 24, CLB_YQ_D_S2.CLB_YQ_D_S2,
CLB_YQ_D_S2.Y2);
        jbits.setTileBits(40, 24, CLB_YQ_RESET_S0.CLB_YQ_RESET_S0,
CLB_YQ_RESET_S0.CLB_SRFF_S0);
        jbits.setTileBits(40, 24, CLB_YQ_RESET_S1.CLB_YQ_RESET_S1,
CLB_YQ_RESET_S1.CLB_SRFF_S1);
        jbits.setTileBits(40, 24, CLB_YQ_RESET_S2.CLB_YQ_RESET_S2,
CLB_YQ_RESET_S2.CLB_SRFF_S2);
        jbits.setTileBits(40, 24, CLB_YQ_SET_S0.CLB_YQ_SET_S0,
CLB_YQ_SET_S0.CLB_REVFF_S0);
        jbits.setTileBits(40, 24, CLB_YQ_SET_S1.CLB_YQ_SET_S1,
CLB_YQ_SET_S1.CLB_REVFF_S1);
        jbits.setTileBits(40, 24, CLB_YQ_SET_S2.CLB_YQ_SET_S2,
CLB_YQ_SET_S2.CLB_REVFF_S2);
        jbits.setTileBits(40, 24, CLK0.CLK0, CLK0.N6D6);
        jbits.setTileBits(40, 24, CLK1.CLK1, CLK1.N6D6);
        jbits.setTileBits(40, 24, CLK2.CLK2, CLK2.N6D6);
        jbits.setTileBits(40, 24, E2BEG8.E2BEG8, E2BEG8.Y3);
        jbits.setTileBits(40, 24, E2BEG9.E2BEG9, E2BEG9.OMUX15);
        jbits.setTileBits(40, 24, F1_B0.F1_B0, F1_B0.BY2);
        jbits.setTileBits(40, 24, F1_B1.F1_B1, F1_B1.S2BEG0);
        jbits.setTileBits(40, 24, F1_B2.F1_B2, F1_B2.BY2);
```

```
            jbits.setTileBits(40, 24, F2_B0.F2_B0, F2_B0.OMUX6);
            jbits.setTileBits(40, 24, F2_B1.F2_B1, F2_B1.S2MID5);
            jbits.setTileBits(40, 24, F2_B2.F2_B2, F2_B2.OMUX_S4);
            jbits.setTileBits(40, 24, F3_B0.F3_B0, F3_B0.S2MID5);
            jbits.setTileBits(40, 24, F3_B1.F3_B1, F3_B1.W2END4);
            jbits.setTileBits(40, 24, F3_B2.F3_B2, F3_B2.OMUX9);
            jbits.setTileBits(40, 24, F4_B0.F4_B0, F4_B0.OMUX2);
            jbits.setTileBits(40, 24, F4_B1.F4_B1, F4_B1.OMUX13);
            jbits.setTileBits(40, 24, F4_B2.F4_B2, F4_B2.BY1);
            jbits.setTileBits(40, 24, G1_B0.G1_B0, G1_B0.BY2);
            jbits.setTileBits(40, 24, G1_B1.G1_B1, G1_B1.OMUX2);
            jbits.setTileBits(40, 24, G1_B2.G1_B2, G1_B2.BY3);
            jbits.setTileBits(40, 24, G2_B0.G2_B0, G2_B0.OMUX6);
            jbits.setTileBits(40, 24, G2_B1.G2_B1, G2_B1.S2MID5);
            jbits.setTileBits(40, 24, G2_B2.G2_B2, G2_B2.W2END4);
            jbits.setTileBits(40, 24, G3_B0.G3_B0, G3_B0.S2MID5);
            jbits.setTileBits(40, 24, G3_B1.G3_B1, G3_B1.W2END4);
            jbits.setTileBits(40, 24, G3_B2.G3_B2, G3_B2.OMUX9);
            jbits.setTileBits(40, 24, G3_B3.G3_B3, G3_B3.W2END2);
            jbits.setTileBits(40, 24, G4_B0.G4_B0, G4_B0.BY0);
            jbits.setTileBits(40, 24, G4_B1.G4_B1, G4_B1.OMUX13);
            jbits.setTileBits(40, 24, G4_B2.G4_B2, G4_B2.BY1);
            jbits.setTileBits(40, 24, G4_B3.G4_B3, G4_B3.W2END9);
            jbits.setTileBits(40, 24, N2BEG6.N2BEG6, N2BEG6.W2END4);
            jbits.setTileBits(40, 24, N2BEG7.N2BEG7, N2BEG7.OMUX11);
            jbits.setTileBits(40, 24, N2BEG8.N2BEG8, N2BEG8.Y3);
            jbits.setTileBits(40, 24, N6BEG6.N6BEG6, N6BEG6.LV0);
            jbits.setTileBits(40, 24, OMUX0.OMUX0, OMUX0.YQ2);
            jbits.setTileBits(40, 24, OMUX11.OMUX11, OMUX11.YQ0);
            jbits.setTileBits(40, 24, OMUX13.OMUX13, OMUX13.XQ1);
            jbits.setTileBits(40, 24, OMUX15.OMUX15, OMUX15.Y3);
            jbits.setTileBits(40, 24, OMUX2.OMUX2, OMUX2.YQ1);
            jbits.setTileBits(40, 24, OMUX3.OMUX3, OMUX3.YQ0);
            jbits.setTileBits(40, 24, OMUX5.OMUX5, OMUX5.YQ1);
            jbits.setTileBits(40, 24, OMUX6.OMUX6, OMUX6.XQ0);
            jbits.setTileBits(40, 24, OMUX9.OMUX9, OMUX9.XQ2);
            jbits.setTileBits(40, 24, S2BEG0.S2BEG0, S2BEG0.OMUX0);
            jbits.setTileBits(40, 24, S2BEG2.S2BEG2, S2BEG2.OMUX6);
            jbits.setTileBits(40, 24, S6BEG6.S6BEG6, S6BEG6.LV0);

        } catch (ConfigurationException ce) {
            System.out.println("Problem modifying the bitstream." + ce);
        }
    }
}
```

B.5 RAISE_Bterms.java

```
// This class configures the bottom Term tiles in the device. This
// makes it possible to send and receive signals through the
// corresponding Input/Output Blocks.

import com.xilinx.JBits.ArchIndependent.ConfigurationException;
import com.xilinx.JBits.Virtex2.*;
import com.xilinx.JBits.Virtex2.Bits.Logic.Bterm.*;
```

```
import com.xilinx.JBits.Virtex2.Bits.Logic.IOIs.*;
import com.xilinx.JBits.Virtex2.Bits.Wires.Bterm.*;

public class RAISE_Bterms {
  public static void configure(JBits jbits) {

    try {
      // Configuration for the Bterm at (0, 2)
      jbits.setTileBits(0, 2, PULL.CONFIG[0], PULL.PULLUP);
      jbits.setTileBits(0, 2, PULL.CONFIG[1], PULL.PULLUP);
      jbits.setTileBits(0, 2, PULL.CONFIG[2], PULL.PULLUP);
      jbits.setTileBits(0, 2, N6END9.N6END9, N6END9.S6D9);

    } catch (ConfigurationException ce) {
      System.out.println("Problem modifying the bitstream." + ce);
    }
  }
}
```

B.6 RAISE_Lterms.java

```
// This class configures the left Term tiles in the device. This
// makes it possible to send and receive signals through the
// corresponding Input/Output Blocks.

import com.xilinx.JBits.ArchIndependent.ConfigurationException;
import com.xilinx.JBits.Virtex2.*;
import com.xilinx.JBits.Virtex2.Bits.Logic.Lterm.*;
import com.xilinx.JBits.Virtex2.Bits.Logic.IOIs.*;
import com.xilinx.JBits.Virtex2.Bits.Wires.Lterm.*;

public class RAISE_Lterms {
  public static void configure(JBits jbits) {

    try {
      // Configuration for the Lterm at (32, 0)
      jbits.setTileBits(32, 0, ENABLE_TS.CONFIG[1], ENABLE_TS.OFF);
      jbits.setTileBits(32, 0, PULL.CONFIG[0], PULL.PULLUP);
      jbits.setTileBits(32, 0, PULL.CONFIG[1], PULL.OFF);
      jbits.setTileBits(32, 0, PULL.CONFIG[2], PULL.PULLUP);

    } catch (ConfigurationException ce) {
      System.out.println("Problem modifying the bitstream." + ce);
    }
  }
}
```

B.7 RAISE_Tterms.java

```
// This class configures the left Term tiles in the device. This
// makes it possible to send and receive signals through the
// corresponding Input/Output Blocks.
```

```
import com.xilinx.JBits.ArchIndependent.ConfigurationException;
import com.xilinx.JBits.Virtex2.*;
import com.xilinx.JBits.Virtex2.Bits.Logic.Tterm.*;
import com.xilinx.JBits.Virtex2.Bits.Logic.IOIs.*;
import com.xilinx.JBits.Virtex2.Bits.Wires.Tterm.*;

public class RAISE_Tterms {
  public static void configure(JBits jbits) {

    try {
      // Configuration for the Tterm at (50, 10)
      jbits.setTileBits(50, 10, ENABLE_TS.CONFIG[1], ENABLE_TS.OFF);
      jbits.setTileBits(50, 10, PULL.CONFIG[0], PULL.PULLUP);
      jbits.setTileBits(50, 10, PULL.CONFIG[1], PULL.OFF);
      jbits.setTileBits(50, 10, PULL.CONFIG[2], PULL.OFF);
      jbits.setTileBits(50, 10, TopIoStandard.INPUT[2],
IoStandard.Input.LVTTL);
      jbits.setTileBits(50, 10, TopIoStandard.OUTPUT[2],
IoStandard.Output.S_OFF_0_X);

      // Configuration for the Tterm at (50, 21)
      jbits.setTileBits(50, 21, PULL.CONFIG[0], PULL.OFF);
      jbits.setTileBits(50, 21, PULL.CONFIG[1], PULL.PULLUP);
      jbits.setTileBits(50, 21, PULL.CONFIG[2], PULL.PULLUP);
      jbits.setTileBits(50, 21, TopIoStandard.INPUT[0],
IoStandard.Input.LVTTL);
      jbits.setTileBits(50, 21, TopIoStandard.OUTPUT[0],
IoStandard.Output.S_OFF_0_X);

    } catch (ConfigurationException ce) {
      System.out.println("Problem modifying the bitstream." + ce);
    }
  }
}
```

Appendix C: The Waterjugs GUI (Java/SWT)

The following code creates the graphical user interface to the serial port and the FPGA. It displays the Soar rules for the Waterjugs algorithm and the JBits code for the FPGA. Users can initialize the jugs' volumes and choose the desired volume to end the search.

This code has two classes. The first, `Raise`, sets up the SWT graphics and provides overall control of the application. The second, `SPort`, enables direct control of the RS-232 serial port.

C.1 Raise.java

```
// This class initializes the Waterjug parameters and displays results
// from the FPGA. It doesn't access the serial port directly, but
// invokes methods from the SPort class in section B.2.

package com.hoplite.raise;
```

```
import java.io.*;

import org.eclipse.swt.*;
import org.eclipse.swt.widgets.*;
import org.eclipse.swt.custom.StyledText;
import org.eclipse.swt.events.*;
import org.eclipse.swt.program.*;

public class Raise extends Composite {

  static Button prog, Send, verb, Stage1, Stage2, Stage3, Stage4;
  static TabFolder tf;
  static StyledText soarText, jbitsText, responseText;
  static Text text1, text2, text3;
  static int vol1, vol2, desired, reps, place;
  static boolean found, pressed, verbose;
  static String[] op = new String[]
  {"Fill J1", "Fill J2", "Pour J1 to J2", "Pour J2 to J1",
   "Empty J1", "Empty J2"};

  public Raise(Composite parent) throws IOException {

    super(parent, SWT.NULL);
    this.setSize(530, 525);
    SPort.portInit();

    String soarString = "";
    String jbitsString = "";
    String responseString = "";
    String textLine;

    found = false;
    verbose = false;

    Group comp = new Group(this, SWT.SHADOW_ETCHED_OUT);
    comp.setText("WMEs");
    comp.setBounds(400, 14, 130, 525);

    Label jug1 = new Label(comp, SWT.LEFT);
    jug1.setText("Jug 1:");
    jug1.setBounds(30, 30, 35, 30);

    text1 = new Text(comp, SWT.RIGHT);
    text1.setBounds(65, 30, 30, 15);
    text1.setText("5");

    Label jug2 = new Label(comp, SWT.LEFT);
    jug2.setText("Jug 2:");
    jug2.setBounds(30, 65, 35, 30);

    text2 = new Text(comp, SWT.RIGHT);
    text2.setBounds(65, 65, 30, 15);
    text2.setText("3");

    Label des = new Label(comp, SWT.LEFT);
    des.setText("Goal:");
    des.setBounds(30, 100, 35, 30);
```

```
text3 = new Text(comp, SWT.RIGHT);
text3.setBounds(65, 100, 30, 15);
text3.setText("1");

Button set = new Button(comp, SWT.PUSH);
set.setBounds(25, 135, 40, 25);
set.setText("Set");
set.addMouseListener(new MouseAdapter () {
  public void mouseDown(MouseEvent e) {
    pressed = true;
    vol1 = Integer.parseInt(text1.getText());
    vol2 = Integer.parseInt(text2.getText());
    desired = Integer.parseInt(text3.getText());
  }
});

Button reset = new Button(comp, SWT.PUSH);
reset.setBounds(65, 135, 40, 25);
reset.setText("Reset");
reset.addMouseListener(new MouseAdapter () {
  public void mouseDown(MouseEvent e) {
    pressed = false;
    prog.setEnabled(true);
    Send.setEnabled(true);
    responseText.setText("");
    for (int i=0; i<11; i++)
      SPort.received[i] = 0;
  }
});

Label sep = new Label(comp, SWT.SEPARATOR | SWT.HORIZONTAL);
sep.setBounds(0, 185, 130, 1);

Label v = new Label(comp, SWT.LEFT);
v.setText("Verbose: ");
v.setBounds(33, 200, 43, 30);

verb = new Button(comp, SWT.CHECK);
verb.setBounds(80, 201, 14, 14);
verb.addSelectionListener(new SelectionAdapter() {
  public void widgetSelected(SelectionEvent arg0) {
    verbose = verb.getSelection();
  }
});

tf = new TabFolder(this, SWT.NONE);
tf.setBounds(0, 0, 400, 525);
TabItem soar = new TabItem(tf, SWT.NONE, 0);
TabItem jbits = new TabItem(tf, SWT.NONE, 1);
TabItem response = new TabItem(tf, SWT.NONE, 2);

soar.setText("SOAR Rules");
soarText = new StyledText(tf, SWT.V_SCROLL);
soar.setControl(soarText);
BufferedReader br
  = new BufferedReader(new FileReader("water.soar"));
```

```
textLine = br.readLine();
while (!(textLine == null)) {
  soarString = soarString + textLine + "\n";
  textLine = br.readLine();
}
br.close();
soarText.setText(soarString);

jbits.setText("JBits Code");
jbitsText = new StyledText(tf, SWT.V_SCROLL);
jbits.setControl(jbitsText);
BufferedReader jbr
  = new BufferedReader(new FileReader("water.soar"));
textLine = jbr.readLine();
while (!(textLine == null)) {
  jbitsString = jbitsString + textLine + "\n";
  textLine = jbr.readLine();
}
jbitsText.setText(jbitsString);
response.setText("FPGA Interface");
responseText = new StyledText(tf, SWT.V_SCROLL);
response.setControl(responseText);
responseText.setText("STAGES OF WATERJUG CALCULATION\n\n");

prog = new Button(comp, SWT.PUSH);
prog.setBounds(20, 235, 90, 30);
prog.setText("Program FPGA");
prog.addMouseListener(new MouseAdapter () {
  public void mouseDown(MouseEvent e) {
    tf.setSelection(1);
    Program.launch("conf.bat");
    prog.setEnabled(false);
    Send.setEnabled(true);
  }
});

Send = new Button(comp, SWT.PUSH);
Send.setBounds(20, 280, 90, 30);
Send.setText("Send WMEs");
Send.setEnabled(false);
Send.addMouseListener(new MouseAdapter () {
  public void mouseDown(MouseEvent e) {
    Send.setEnabled(false);
    if (!pressed) {
      String soarCode = Raise.soarText.getText();
      int place1 = soarCode.indexOf("volume");
      vol1 = Integer.parseInt(
        soarCode.substring(place1+7, place1+8));
      int place2 = soarCode.indexOf("volume", place1+1);
      vol2 = Integer.parseInt(
        soarCode.substring(place2+7, place2+8));
      int place3 = soarCode.lastIndexOf("contents");
      desired = Integer.parseInt(
        soarCode.substring(place3+9, place3+10));
    }
    responseText.append(String.valueOf(vol1));
    responseText.append(String.valueOf(vol2));
```

```
      responseText.append(String.valueOf(desired));
      int val = vol2 * 16 + vol1;
      try {
        SPort.sp.putByte((byte)val);
        for(int i=0; i<50000000; i++);
        SPort.sp.putByte((byte)desired);
      } catch (IOException e1) {
        System.out.println("Didn't send");
      }
      Stage1.setEnabled(true);
      prog.setEnabled(false);
      SPort.rcv.start();
    }
  });

  Stage1 = new Button(comp, SWT.PUSH);
  Stage1.setBounds(20, 325, 90, 30);
  Stage1.setText("Stage 1");
  Stage1.setEnabled(false);
  Stage1.addMouseListener(new MouseAdapter () {
    public void mouseDown(MouseEvent e) {
      tf.setSelection(2);
      Stage1.setEnabled(false);
      try {
        SPort.sp.putByte((byte)255);
      } catch (IOException e1) {
        System.out.println("Problem with Stage.");
      }
      responseText.append("Stage 1:\n\n");
      if (verbose) {
        responseText.append(op[0]+": "+(vol1==desired)+"\n");
        responseText.append(op[1]+": "+(vol1==desired)+"\n");
        responseText.append(op[2]+": "+(vol1==desired)+"\n");
        responseText.append(op[3]+": "+(vol1==desired)+"\n");
        responseText.append(op[4]+": "+(desired==0)+"\n");
        responseText.append(op[5]+": "+(vol1==desired)+"\n");
      }
      found = ((vol1==desired) || (desired == 0));
      responseText.append(found ? "The goal has been reached.\n\n" :
        "The goal has not been reached.\n\n");
      Stage2.setEnabled(!found);
      while(!(SPort.got));
      String ans;
      for (int i=0; i<11; i++)
        responseText.append(String.valueOf(SPort.received[i])+"\n");
      responseText.append("\n");
    }
  });

  Stage2 = new Button(comp, SWT.PUSH);
  Stage2.setBounds(20, 370, 90, 30);
  Stage2.setText("Stage 2");
  Stage2.setEnabled(false);
  Stage2.addMouseListener(new MouseAdapter () {
    public void mouseDown(MouseEvent e) {
      Stage2.setEnabled(false);
      responseText.append("\nStage 2:\n\n");
```

```
    if (verbose) {
      for(int i=0; i<4; i++)
        for(int j=0; j<6; j++)
          responseText.append(op[i]+", "+op[j]+":
            "+(vol1==desired)+"\n");
      for(int j=0; j<3; j++)
        responseText.append(op[4]+", "+op[j]+":
          "+(vol1==desired)+"\n");
      responseText.append(op[4]+", "+op[3]+": "+
      ((SPort.received[0] & (1<<3))!=0)+"\n");
      for(int j=4; j<6; j++)
        responseText.append(op[4]+", "+op[j]+":
          "+(vol1==desired)+"\n");
      for(int i=0; i<2; i++)
        responseText.append(op[5]+", "+op[i]+":
          "+(vol1==desired)+"\n");
      responseText.append(op[5]+", "+op[2]+": "+
        ((SPort.received[0] & (1<<2))!=0)+"\n");
      for(int i=3; i<6; i++)
        responseText.append(op[5]+", "+op[i]+":
          "+(vol1==desired)+"\n");
    }
    found = (((SPort.received[0] & (1<<2))!=0) ||
      ((SPort.received[0] & (1<<3))!=0));
    responseText.append(found ? "The goal has been reached.\n\n" :
      "The goal has not been reached.\n\n");
    Stage3.setEnabled(!found);
    if ((SPort.received[0] & (1<<2))!=0)
      responseText.append("Sequence: "+op[4]+", "+op[3]+"\n\n");
    if ((SPort.received[0] & (1<<3))!=0)
      responseText.append("Sequence: "+op[5]+", "+op[2]+"\n\n");
  }
});

Stage3 = new Button(comp, SWT.PUSH);
Stage3.setBounds(20, 415, 90, 30);
Stage3.setText("Stage 3");
Stage3.setEnabled(false);
Stage3.addMouseListener(new MouseAdapter () {
  public void mouseDown(MouseEvent e) {
    Stage3.setEnabled(false);
    responseText.append("\nStage 3:\n\n");
    if (verbose) {
      for(int i=0; i<4; i++) {
        for(int j=0; j<6; j++)
          for(int k=0; k<6; k++)
            responseText.append(op[i]+", "+op[j]+", "+op[k]+":
              "+(vol1==desired)+"\n");
            responseText.append("\n");
      }
      for(int j=0; j<3; j++)
        for(int k=0; k<6; k++)
          responseText.append(op[4]+", "+op[j]+", "+op[k]+":
            "+(vol1==desired)+"\n");
      for(int x=0; x<6; x++)
        responseText.append(op[4]+", "+op[3]+", "+op[x]+":
        "+((SPort.received[1] & (1<<(7-x)))!=0)+"\n");
```

```
        for(int j=4; j<6; j++)
          for(int k=0; k<6; k++)
            responseText.append(op[4]+", "+op[j]+", "+op[k]+":
            "+(vol1==desired)+"\n");
        responseText.append("\n");

        for(int j=0; j<2; j++)
          for(int k=0; k<6; k++)
            responseText.append(op[5]+", "+op[j]+", "+op[k]+":
              "+(vol1==desired)+"\n");
        for(int x=0; x<2; x++)
          responseText.append(op[5]+", "+op[2]+", "+op[x]+":
            "+((SPort.received[1] & (1<<(1-x)))!=0)+"\n");
        for(int x=2; x<6; x++)
          responseText.append(op[5]+", "+op[2]+", "+op[x]+":
            "+((SPort.received[0] & (1<<(9-x)))!=0)+"\n");
        for(int j=3; j<6; j++)
          for(int k=0; k<6; k++)
        responseText.append(op[5]+", "+op[j]+", "+op[k]+":
          "+(vol1==desired)+"\n");
        responseText.append("\n");
      }
      found = ((SPort.received[0] != 0) || (SPort.received[1] != 0));
      responseText.append(found ? "The goal has been reached.\n" :
        "The goal has not been reached.\n\n");
      Stage4.setEnabled(!found);
      for(int i=0; i<2; i++) {
        for (int j=0; j<8; j++)
          if ((SPort.received[i] & (1<<j)) != 0) {
            place = i*8+j-4;
            if (place < 6)
              responseText.append("Sequence: "+op[5]+", "+op[2]+",
              "+op[5-place]+"\n");
            else {
              place = place - 36;
              responseText.append("Sequence: "+op[4]+", "+op[3]+", "+
              op[5-(place-6)]+", "+op[5-place%6]+"\n");
            }
          }
      }
    }
  });

  Stage4 = new Button(comp, SWT.PUSH);
  Stage4.setBounds(20, 460, 90, 30);
  Stage4.setText("Stage 4");
  Stage4.setEnabled(false);
  Stage4.addMouseListener(new MouseAdapter () {
    public void mouseDown(MouseEvent e) {
      Stage4.setEnabled(false);
      responseText.append("\nStage 4:\n\n");
      if (verbose) {
        for(int i=0; i<4; i++) {
          for(int j=0; j<6; j++)
            for(int k=0; k<6; k++)
              for(int l=0; l<6; l++)
                responseText.append(op[i]+", "+op[j]+", "+op[k]+",
```

```
            "+op[l]+": "+(vol1==desired)+"\n");
        responseText.append("\n");
    }
    for(int j=0; j<3; j++)
      for(int k=0; k<6; k++)
        for(int l=0; l<6; l++)
          responseText.append(op[4]+", "+op[j]+", "+op[k]+",
            "+op[l]+": "+(vol1==desired)+"\n");
    for(int x=0; x<6; x++)
      responseText.append(op[4]+", "+op[3]+", "+op[0]+",
        "+op[x]+": "+((SPort.received[10] & (1<<(7-x)))!=0)
          +"\n");
    for(int x=0; x<2; x++)
      responseText.append(op[4]+", "+op[3]+", "+op[1]+",
        "+op[x]+": "+((SPort.received[10] & (1<<(1-x)))!=0)
          +"\n");
    for(int x=2; x<6; x++)
      responseText.append(op[4]+", "+op[3]+", "+op[1]+",
      "+op[x]+": "+((SPort.received[9] & (1<<(9-x)))!=0)+"\n");
    for(int x=0; x<4; x++)
      responseText.append(op[4]+", "+op[3]+", "+op[2]+",
        "+op[x]+": "+((SPort.received[9] & (1<<(3-x)))!=0)+"\n");
    for(int x=4; x<6; x++)
      responseText.append(op[4]+", "+op[3]+", "+op[2]+",
        "+op[x]+": "+((SPort.received[8] & (1<<(11-x)))!=0)
          +"\n");
    for(int x=0; x<6; x++)
      responseText.append(op[4]+", "+op[3]+", "+op[3]+",
        "+op[x]+": "+((SPort.received[8] & (1<<(5-x)))!=0)+"\n");
    for(int x=0; x<6; x++)
      responseText.append(op[4]+", "+op[3]+", "+op[4]+",
        "+op[x]+": "+((SPort.received[7] & (1<<(7-x)))!=0)+"\n");
    for(int x=0; x<2; x++)
      responseText.append(op[4]+", "+op[3]+", "+op[5]+",
        "+op[x]+": "+((SPort.received[7] & (1<<(1-x)))!=0)+"\n");
    for(int x=2; x<6; x++)
      responseText.append(op[4]+", "+op[3]+", "+op[5]+",
        "+op[x]+": "+((SPort.received[6] & (1<<(9-x)))!=0)+"\n");
    for(int j=4; j<6; j++)
      for(int k=0; k<6; k++)
        for(int l=0; l<6; l++)
          responseText.append(op[4]+", "+op[j]+", "+op[k]+",
            "+op[l]+": "+(vol1==desired)+"\n");
    responseText.append("\n");
    for(int j=0; j<2; j++)
      for(int k=0; k<6; k++)
        for(int l=0; l<6; l++)
          responseText.append(op[5]+", "+op[j]+", "+op[k]+",
            "+op[l]+": "+(vol1==desired)+"\n");
    for(int x=0; x<4; x++)
      responseText.append(op[5]+", "+op[2]+", "+op[0]+",
        "+op[x]+": "+((SPort.received[6] & (1<<(3-x)))!=0)+"\n");
    for(int x=4; x<6; x++)
      responseText.append(op[5]+", "+op[2]+", "+op[0]+",
        "+op[x]+": "+((SPort.received[5] & (1<<(11-x)))!=0)
          +"\n");
    for(int x=0; x<6; x++)
```

```
            responseText.append(op[5]+", "+op[2]+", "+op[1]+",
              "+op[x]+": "+((SPort.received[5] & (1<<(5-x)))!=0)+"\n");
          for(int x=0; x<6; x++)
            responseText.append(op[5]+", "+op[2]+", "+op[2]+",
              "+op[x]+": "+((SPort.received[4] & (1<<(7-x)))!=0)+"\n");
          for(int x=0; x<2; x++)
            responseText.append(op[5]+", "+op[2]+", "+op[3]+",
              "+op[x]+": "+((SPort.received[4] & (1<<(1-x)))!=0)+"\n");
          for(int x=2; x<6; x++)
            responseText.append(op[5]+", "+op[2]+", "+op[3]+",
              "+op[x]+": "+((SPort.received[3] & (1<<(9-x)))!=0)+"\n");
          for(int x=0; x<4; x++)
            responseText.append(op[5]+", "+op[2]+", "+op[4]+",
              "+op[x]+": "+((SPort.received[3] & (1<<(3-x)))!=0)+"\n");
          for(int x=4; x<6; x++)
            responseText.append(op[5]+", "+op[2]+", "+op[4]+",
              "+op[x]+": "+((SPort.received[2] & (1<<(11-x)))!=0)
                +"\n");
          for(int x=0; x<6; x++)
            responseText.append(op[5]+", "+op[2]+", "+op[5]+",
              "+op[x]+": "+((SPort.received[2] & (1<<(5-x)))!=0)+"\n");
          for(int j=3; j<6; j++)
            for(int k=0; k<6; k++)
              for(int l=0; l<6; l++)
                responseText.append(op[5]+", "+op[j]+", "+op[k]+",
                  "+op[l]+": "+(vol1==desired)+"\n");
          responseText.append("\n");
        }
        for(int i=2; i<11; i++)
          if (SPort.received[i] != 0)
            found = true;
        responseText.append(found ? "The goal has been reached.\n" :
          "The goal has not been reached.\n\n");
        for(int i=2; i<11; i++) {
          for (int j=0; j<8; j++)
            if ((SPort.received[i] & (1<<j)) != 0) {
              place = (i-2)*8 + j;
              if (place < 36)
                responseText.append("Sequence: "+op[5]+", "+op[2]+",
                  "+op[5-place/6]+", "+op[5-place%6]+"\n");
              else {
                place = place - 36;
                responseText.append("Sequence: "+op[4]+", "+op[3]+",
                  "+op[5-place/6]+", "+op[5-place%6]+"\n");
              }
            }
        }
      }
    });
  }

  public static void main (String [] args) throws IOException {

    Display display = new Display();
    Shell shell = new Shell(display, SWT.DIALOG_TRIM);
    shell.setText("FPGA-based Demonstration of Soar Waterjugs
      Algorithm");
```

```
    shell.setLocation(90, 15);
    Raise r = new Raise(shell);

    shell.open();
    shell.pack();
    while (!shell.isDisposed()) {
      if (!display.readAndDispatch())
        display.sleep();
    }
    display.dispose();
    SPort.sp.close();
  }
}
```

C.2 SPort.java

```
// This class interfaces with the RS-232 port using the proprietary
// Serialio package.  This makes it possible to send and receive data
// from the FPGA.

package com.hoplite.raise;

import java.io.FileInputStream;
import java.io.IOException;
import java.io.ObjectInputStream;

import Serialio.*;

public class SPort {

  static SerialConfig serCfg;
  static SerialPortLocal sp;
  static TermRcvTask rcv;
  static TermSndTask snd;
  static int[] received = new int[11];
  static boolean got = false;

  static void portInit() throws IOException {
    try {
      serCfg = readLastConfig();
     } catch (Exception cnf) {}

    String devName = new String(serCfg.getPortName());
    sp = new SerialPortLocal(serCfg);
    sp.setDTR(true);
    rcv = new TermRcvTask(sp);
    snd = new TermSndTask(sp);
  }

  static SerialConfig readLastConfig() throws Exception {
    FileInputStream fs = new FileInputStream("LastConfig.ser");
    ObjectInputStream is = new ObjectInputStream(fs);
    SerialConfig cfg = (SerialConfig)is.readObject();
    is.close();
    fs.close();
```

```
    return cfg;
  }

  static class TermRcvTask extends Thread {

    SerialPortLocal p;
    TermRcvTask(SerialPortLocal sp) throws IOException
    {
      if (sp.getPortNum() == -1)
        throw new IOException("TermSndTask: serial port not
          initialized");
      p = sp;
    }

    public void run() {
      int b = -1;
      int count = 0;
      try {
        while (count < 11) {
          while((b = p.getByte()) != -1) {
            received[count] = b;
            count = count + 1;
          }
        }
        got = true;
        try {Thread.sleep(100);} catch (InterruptedException e){}
      } catch (Exception e) {
        System.out.println("Error in TermRcvTask "+e);
      }
    }
  }

  static class TermSndTask extends Thread {

    SerialPortLocal p;
    TermSndTask(SerialPortLocal sp) throws IOException {
      if (sp.getPortNum() == -1)
      throw new IOException("TermRcvTask: serial port
        not initialized");
      p = sp;
    }

    public void run() {
      int first = Raise.vol1*16 + Raise.vol2;
      int second = Raise.desired;
      try {
        p.putByte((byte)first);
        p.putByte((byte)second);
      } catch (Exception e) {
        System.out.println("Error in TermSndTask "+e);
      }
    }
  }
}
```